

NAVAL POSTGRADUATE SCHOOL

Monterey, California



FORM QUALITY INSPECTED 4

THESIS

**DESIGN AND IMPLEMENTATION OF AN ENTERPRISE
INFORMATION SYSTEM UTILIZING A COMPONENT BASED
THREE-TIER CLIENT/SERVER DATABASE SYSTEM**

by

Murat Akbay
Steven C. Lewis

March 1999

Thesis Advisor:
Second Reader:

C. Thomas Wu
Chris Eagle

Approved for public release; distribution is unlimited.

1 999021 9 08 9

Preceding Pages Blank

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.

1. AGENCY USE ONLY (Leave blank)

2. REPORT DATE
March 1999

3. REPORT TYPE AND DATES COVERED
Master's Thesis

4. TITLE AND SUBTITLE: DESIGN AND IMPLEMENTATION OF AN ENTERPRISE INFORMATION SYSTEM UTILIZING A COMPONENT BASED THREE-TIER CLIENT/SERVER DATABASE SYSTEM

5. FUNDING NUMBERS

6. AUTHOR(S) Akbay, Murat, Lewis, Steven C.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION
REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING / MONITORING
AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

The views expressed in this thesis are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (maximum 200 words)

The Naval Security Group currently requires a modern architecture to merge existing command databases into a single Enterprise Information System through which each command may manipulate administrative data. There are numerous technologies available to build and implement such a system. Component-based architectures are extremely well-suited for creating scalable and flexible three-tier Client/Server systems because the data and business logic are encapsulated within objects, allowing them to be located anywhere on a network. The first tier represents the visual aspects of the data on the client side. The middle tier consists of server objects that represent the persistent data and enforce the business logic functions. The third tier maintains the database management systems. The client interacts with the middle-tier server objects via Common Object Request Broker Architecture. CORBA provides a language and platform independent architecture that enables objects to transparently make requests and receive responses in a distributed environment. Java is an object-oriented, multi-threaded, secure mobile code system that allows applications to run on all major computing platforms. This thesis examines the design of an EIS using Java Applets that use Inter-Orb Protocol to communicate with CORBA middle-tier server objects. The third tier will incorporate Java Database Connectivity to communicate with database management systems.

14. SUBJECT TERMS Enterprise Information System, Naval Security Group, Database, JDBC, Java, CORBA, Client/Server, SQL Server

15. NUMBER OF PAGES
293

16. PRICE CODE

17. SECURITY CLASSIFICATION
OF REPORT
Unclassified

18. SECURITY CLASSIFICATION
OF THIS PAGE
Unclassified

19. SECURITY CLASSIFI-
CATION OF ABSTRACT
Unclassified

20. LIMITATION OF ABSTRACT
UL

Approved for public release; distribution is unlimited

**DESIGN AND IMPLEMENTATION OF
AN ENTERPRISE INFORMATION SYSTEM UTILIZING A COMPONENT BASED THREE-TIER
CLIENT/SERVER DATABASE SYSTEM**

Murat Akbay
First Lieutenant, Turkish Army
B.S., Turkish Military Academy, 1991

Steven C. Lewis
Lieutenant, United States Navy
B.S., Oregon State University, 1993

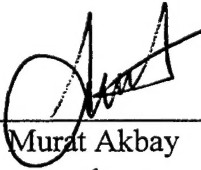
Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

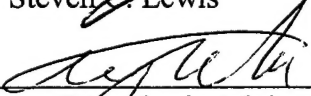
**NAVAL POSTGRADUATE SCHOOL
March 1999**

Authors:

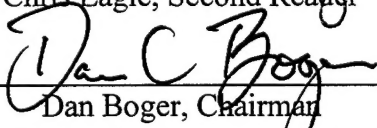

Murat Akbay


Steven C. Lewis

Approved by:


C. Thomas Wu, Thesis Advisor


Chris Eagle, Second Reader


Dan Boger, Chairman
Department of Computer Science

ABSTRACT

The Naval Security Group currently requires a modern architecture to merge existing command databases into a single Enterprise Information System through which each command may manipulate administrative data. There are numerous technologies available to build and implement such a system. Component-based architectures are extremely well-suited for creating scalable and flexible three-tier Client/Server systems, because the data and business logic are encapsulated within objects, allowing them to be located anywhere on a network. The first tier represents the visual aspects of the data on the client side. The middle tier consists of server objects that represent the persistent data and enforce the business logic functions. The third tier maintains the database management systems. The client interacts with the middle-tier server objects via Common Object Request Broker Architecture. CORBA provides a language and platform independent architecture that enables objects to transparently make requests and receive responses in a distributed environment. Java is an object-oriented, multi-threaded, secure mobile code system that allows applications to run on all major computing platforms. This thesis examines the design of an EIS using Java applets that use Inter-Orb Protocol to communicate with CORBA middle-tier server objects. The third tier will incorporate Java Database Connectivity to communicate with database management systems.

TABLE OF CONTENTS

I. INTRODUCTION	1
A. BACKGROUND	1
B. SCOPE	2
C. ORGANIZATION OF THESIS	2
II. ENTERPRISE INFORMATION SYSTEMS.....	5
1. Hardware	7
2. Software	9
3. Interface	10
4. Telecommunications	11
III. RELATIONAL DATABASES AND THE STRUCTURED QUERY LANGUAGE.....	13
1. Data Definition Language	14
2. Data Manipulation Language	15
IV. CLIENT/SERVER ARCHITECTURE	21
A. COMPONENT BASED THREE-TIER CLIENT/SERVER SYSTEMS	25
V. JAVA AND JDBC	27
A. JAVA	27
B. JDBC	29
1. Establishing a connection with the database	30
2. Sending SQL statements	31
3. Processing the results	31
C. JDBC INTERFACES	32
1. DriverManager	33
2. Connection Interface	33
3. Statement Interface	33
4. PreparedStatement Interface.....	34
5. CallableStatement Interface.....	34
6. ResultSet Interface	34
D. JDBC AND CLIENT/SERVER MODELS	35
E. JDBC DRIVERS.....	37
1. JDBC-ODBC bridge plus ODBC Driver	38
2. Native-API partly-Java Driver	39
3. JDBC-Net pure Java Driver	39
4. Native-protocol pure Java Driver	39

F. JDBC SUMMARY.....	39
VI. COMMON OBJECT REQUEST BROKER ARCHITECTURE	41
A. OBJECT MANAGEMENT ARCHITECTURE.....	42
a. Object Request Broker(ORB).....	43
b. CORBA Services.....	45
c. CORBA Facilities.....	48
d. CORBA Domains.....	49
e. Application Objects	49
B. INTERNET INTER-ORB PROTOCOL.....	50
C. INTERFACE DEFINITION LANGUAGE.....	50
a. The module	51
b. Primitive Data Types.....	52
c. Constructed Types	52
d. Container Types	54
e. Interfaces	54
f. Exceptions.....	56
g. Other constructs.....	57
D. CORBA AND THE WEB.....	59
VII. IMPLEMENTATION OF A WEB BASED CLIENT/SERVER SYSTEM USING CORBA AND JDBC	61
A. ORB SELECTION.....	61
B. APPLICATION DESIGN PROCESS WITH VISIBROKER	61
1) Make an analysis of the functionality required of the system and divide the functionality into objects	61
2) Compile the IDL file with an IDL compiler to create the client stub code and server skeleton code in the desired implementation language.....	62
3) The compiler creates a number of files that are used by the ORB to process remote method invocations	62
4) Write the implementation for the server	64
5) Write the client implementation that utilizes the services provided by the server object.....	67
6) Compile the client and the server code with the language compiler.....	68
7) Start the server	68
8) Start the client application.....	69
C. APPLETS AND CORBA.....	70
D. CORBA NAMING SERVICE.....	73
E. DESIGN AND IMPLEMENTATION OF THE EIS PROTOTYPE.....	77
1. Design.....	77
2. Implementation.....	81
3. Data Server	82

4. CORBA Server	86
5. Applet Client	92
F. SYSTEM OPERATION SCENARIO	99
VIII. CONCLUSION	105
A. SYNOPSIS AND CONCLUSION	105
B. SECURITY	106
C. AREAS FOR FURTHER RESEARCH	107
LIST OF REFERENCES	109
APPENDIX A. UML NOTATION REFERENCE TABLE	111
APPENDIX B. NAVSEC.IDL FILE	113
APPENDIX C. SERVER FILES	123
APPENDIX D. CLIENT FILES	211
INITIAL DISTRIBUTION LIST	275

LIST OF FIGURES

Figure 4.1: Two-Tiered Client/Server model	23
Figure 4.2: Three-Tiered Client/Server model	24
Figure 4.3: N-Tiered Client/Server model	25
Figure 5.1: Typical Java environment.....	29
Figure 5.2: Diagram of all JDBC interfaces and classes	32
Figure 5.3: JDBC Two-Tier model	36
Figure 5.4: JDBC Three-Tier model	37
Figure 5.5: JDBC driver implementation	38
Figure 6.1: Object management architecture.....	43
Figure 6.2: The structure of a CORBA 2.0 ORB	44
Figure 6.3: CORBA Services in the object management architecture	46
Figure 6.4: Reuse of OMG specifications	49
Figure 6.5: The structure of an IDL file	51
Figure 7.1: IDL design steps	70
Figure 7.2: Applet-CORBA interaction	72
Figure 7.3: Operation of the Gatekeeper	73
Figure 7.4: How Clients and Servers interact with a Name Server.....	75
Figure 7.5: EIS class diagram.....	79
Figure 7.6: "Add Air Mission" sequence diagram	80
Figure 7.7: Objects in the EIS architecture	82
Figure 7.8: NSGDB Database structure	83
Figure 7.9: GUI class diagram	93
Figure 7.10: AdminPanel class diagram.....	93
Figure 7.11: Prototype log in screenshot.....	101
Figure 7.12: Applet switchboard	101
Figure 7.13: Sample prototype database query screenshot	102
Figure 7.14: Sample prototype query result screenshot	102

LIST OF TABLES

Table 2.1: Characteristics typical of an Enterprise Information System.....	6
Table 2.2: Factors that contribute to the development of an EIS	7
Table 2.3: Types of interfaces available to the EIS.....	10
Table 3.1: Relational Database.....	14
Table 3.2: Comparison of CLI and Embedded SQL features	18
Table 5.1: SQL and Java data types and recommended conversion methods.....	35
Table 6.1: Basic IDL types supported in IDL	52
Table 6.2: Standard IDL exceptions.....	56

ACKNOWLEDGEMENT/DEDICATIONS

We would like to extend our sincere gratitude to our thesis advisors, Professor C. Thomas Wu and LCDR Chris Eagle, for their patience. Additionally, we would like to thank our wives, Neslin Akbay and Tracy Lewis for enduring the entire thesis process. And finally, we would like to dedicate this thesis to our daughters, Ilgin and Katherine for their unconditional love, as well as Murat's mother Ayse Yilmaz, for her unselfish support.

I. INTRODUCTION

A. BACKGROUND

The United States Naval Security Group (NSG), in its present state, lacks the automated information infrastructure required to efficiently process its administrative data. This hinders NSG's ability to rapidly make decisions based on this administrative information. The lack of an adequate information infrastructure results in redundant and imprecise data maintained at different field sites. This ultimately leads to a waste of computer resources, manpower and time. Due to downsizing and budget cuts, our modern Navy is consistently required to "do more with less". An Enterprise Information System is a distributed architecture model that allows organizations to access and manipulate data via a network medium. An Enterprise Information System will greatly benefit the Naval Security Group in manpower and expenditure reductions.

Preliminary studies were performed utilizing a PC-based system, Common Gateway Interface (CGI) scripting and Microsoft products by LCDR James Stevenson [Ref: 1]. From his research, a prototype was designed and built to show the feasibility of implementing an EIS using Commercial Off The Shelf (COTS) hardware and software.

CGI scripting used by [Ref: 1] has serious limitations for implementing a Client/Server EIS. CGI scripts spawn a new process every time a connection request is received. This brings an enormous performance penalty, because there is no way to provide scalability and load balancing with CGI scripting. In addition, CGI scripting is not designed for a full-scale Client/Server application in mind. There are a number of proposed solutions to overcome the inefficiencies of CGI scripting. The latest advances in this field center around component based middleware that makes optimum use of network and hardware resources. The focus of this thesis will be to choose what we deem "the best" of these component based technologies and use it to implement an EIS. From these technologies, a prototype will be designed and built on an open network Client/Server environment. The primary research objective of this research is to provide an object-oriented web-based data retrieval system that can be used as a proof of concept for our proposed NSG EIS.

We will try to answer the following questions:

- What is the most effective and efficient approach to build a “real time” Enterprise Information System, in accordance with IT-21 architecture, that will meet the needs of the Naval Security Group (NAVSECGRU)?
- What are the advantages and disadvantages of accessing a Database via the Internet utilizing Common Object Request Broker Architecture (CORBA) and JDBC?

B. SCOPE

The scope of our thesis is to design and implement a component-based Enterprise Information System prototype for NSG. First we will provide a detailed definition of an EIS. In the second step we will examine various Client/Server configurations to determine which model meets NSG requirements. We will then give the user the appropriate background information for using Java and JDBC as a means of database connectivity. Java will be used as the implementation language for our EIS prototype. Information will be provided about the Common Object Request Broker Architecture and its services, which forms the basis for the interaction between different components in EIS. Then we will begin describing the EIS prototype that we have built. The design of the EIS prototype and implementation issues will be discussed. Next we will set up the hardware and implement the prototype. And finally, we will summarize the lessons learned and recommended future work.

Our EIS prototype will be constructed in accordance with IT-21 specifications and will provide a cost-effective solution for the Naval Security Group automated information infrastructure.

C. ORGANIZATION OF THESIS

This thesis is organized into the following chapters:

- Chapter I: Introduction. This chapter gives an overview of the problem, justification, purpose and basic structure of the thesis.
- Chapter II: Enterprise Information System. This chapter provides the specific concepts of what a modern EIS should be. An explanation of the EIS background history, characteristics, hardware and software will be provided.
- Chapter III: Relational Databases and the Structured Query Language. This chapter describes a Relational Database Management System (RDBMS) and the Structured Query Language (SQL) required to access it. RDBMS and SQL formulate the backbone of our Enterprise Information System prototype.
- Chapter IV: Client/Server Architecture. In this chapter we provide an overview of one-tier, two-tier, three-tier and n-tier Client/Server architectures that can be used in implementing an EIS. These architectures will be explained in detail, to include the advantages and disadvantages of using each. Emphasis will be placed on the three-tier Client/Server system on which our EIS prototype is built.
- Chapter V: Java and JDBC. Java and the JDBC package provide the developers of an EIS a concise and efficient way to access and manipulate data stored in a RDBMS. The interaction between the server and back-end data sources of our EIS prototype is based on JDBC. We have used the methods defined in the JDBC package to execute the queries requested by the user of the EIS on the back-end database server. This chapter will describe how to use Java and JDBC to provide this type of interaction. It will outline the JDBC API, classes, methods, and how they can be used by applications to directly access a RDBMS.
- Chapter VI: Common Object Request Broker Architecture. CORBA is an industry-wide standard for developing robust distributed systems. The interaction between the client and the server components of our EIS prototype is built using the CORBA architecture model. This chapter will provide the user with a broad overview of CORBA, CORBA Services, CORBA Facilities

and why CORBA is an excellent COTS technology for a web-based Client/Server Enterprise Information System.

- Chapter VII: Implementation of a Web Based Client/Server System using CORBA and JDBC. This chapter takes the reader through an application design process with CORBA and then describes the convention of applets as an interface to a CORBA Client/Server system. Next, the design and implementation details of our EIS prototype are discussed and sample diagrams are provided.
- Chapter VIII: Conclusions. This chapter will give directions on how to enhance the prototype that is built and provide the reader with an assessment of the maturity and performance of CORBA in an EIS.

II. ENTERPRISE INFORMATION SYSTEMS

The term Enterprise Information Systems (EIS) or Executive Information Systems was first used at Massachusetts Institute of Technology (MIT) in the late 1970s. The concept spread gradually into dozens of large corporations. A study conducted by the MIT Center for Information Systems Research, revealed that in 1985, about one third of U.S. corporations had some kind of EIS installed or under installation. [Ref: 2] The implementation of an EIS will allow for the reduction of personnel and computer resources required to support multiple queries for each command in the Naval Security Group. In addition, an EIS will eliminate the resources drain on NAVSECGRU systems when executing long-running, complex queries.

EIS utilize newer computer technology in the form of data sources, hardware and programs to place data in a common format and provide fast and easy access to information via the Internet. Our EIS prototype is tailored to the needs and preferences of NAVSECGRU and information is presented in a format that can most readily be interpreted.

Traditionally, Enterprise Information Systems were developed as mainframe computer-based programs. The intent was to present key organizational data for decision-makers that are not well acquainted with computers. The first programs were proprietary and very expensive. They pulled data from mainframe systems and simplified it to graphically illustrate key performance indicators. The objective was to develop computer applications that would address the information needs of senior personnel.

Today's generation of EIS is aimed at a broader audience and the application goes well beyond the boundaries of typical organization hierarchies. EIS are now installed at the personal computer or workstation level on Local Area Networks (LAN). Enterprise Information Systems take advantage of the Client/Server environment, where each person's personal computer has access to organizational data and decides which data are needed to perform their job functions. This arrangement provides the capability for all users to customize their access to the appropriate organizational data and provide relevant

information up the chain of command as well as down, thereby providing timely and up to date data required by decision-makers at the command and fleet levels.

The recent expansion of the World Wide Web (WWW) has enhanced its use as a medium for the dissemination of database information. This is due to the relatively inexpensive and maintenance free aspects of the Internet. Perceivably it makes sense to utilize this medium, thereby decreasing the costs of building and maintaining an EIS.

An EIS has many distinct features that differentiate it from other applications software. A list of these features is presented in Table 2.1. A successful Enterprise Information System minimizes hard copy reports while keeping various branches of the organization up dated.

Characteristics	Description
Degree of use	High, consistent, without need of technical assistance
Computer skills required	Very low --must be easy to learn and use
Flexibility	High
Principle use	Tracking, control, sharing data
Decisions supported	Management
Data supported	Internal and external data
Output capabilities	Text, tabular, graphical
Graphic concentration	High, presentation style
Data access speed	Must be high, fast response time

Table 2.1: Characteristics typical of an Enterprise Information System [Ref: 2]

Advanced internal control and communications are typical focuses of an EIS. An EIS allows access to external as well as internal information as shown in Table 2.2. Both types of data play a vital role in decisions made by users of an EIS. It's the inclusion of these factors, both internal and external to NAVSECGRU that will drive the successful deployment of an EIS in the cryptologic community.

Internal	External
Need for timely information	Need to downsize organizations
Need for improved communication	Rapidly changing operational environment
Need for access to operational data	Need to access external databases
Need for rapid updates from different commands	Need to proactively approach external environment
Need to access command personnel databases	Increasing Government regulations

Table 2.2: Factors that contribute to the development of an EIS [Ref: 2]

With the increasing requirement for information flow at the command level, the importance of Enterprise Information Systems is increasing. An indication of this, is the large expenditure on EIS development projects within the DOD and the subsequent operation of such systems. Initially only large corporate organizations could afford having an EIS. However, as EIS building blocks become cheaper and tools become more readily available, this type of system is becoming affordable for a larger number of organizations.

The latest trend in EIS development is toward the utilization of rapid application development tools to design reusable components. This provides the capability to replace any component of the system when an improved version becomes available. In other words, it is relatively easy to perform a version upgrade to your current database server, or to switch database servers simply by changing only a few lines of code. This will also provide the added advantage of keeping the EIS up to date with advances in technology.

1. Hardware

Since an integrated system requires fairly large amounts of storage space, most EIS originally were developed as mainframe computer solutions. A mainframe EIS requires computer personnel to develop and maintain the system. These systems are very expensive, and their use is usually limited to top-level executives in the organization.

With the advent of Local Area Network technology as well as the expansion of the Internet, several EIS products for networked workstations have become available. These systems have the advantage of requiring less support and less expensive computer hardware. They also increase access of the EIS information to many more users within the organization. EIS have migrated from mainframe computers to personal computers (PC) connected by a LAN and comprised of midrange computers that act as servers to the network. The midrange computer is more powerful than the PC in terms of processing capability and also has the ability to integrate multiple databases. The advantages of the PC are:

- Efficient data summarization
- More user friendly than mainframe computers
- Better graphic capabilities than mainframe computers
- Affordability (the client can be configured less expensively using PC architecture)

The best architecture solution will optimize the capabilities of the PC and the midrange computer. It is truthful to say that the best EIS architecture solution will be built around open-network Client/Server systems. Some of the benefits of the Client/Server architecture are:

- Client/Server systems provide multiple views of data regardless of the data format.
- Client/Server systems reduce investment in computer hardware.
- They establish a flexible system that can change and expand as operational requirements change, responding to a dynamic environment and to the needs of users at all levels.

- They remove barriers across the organization.
- They allow upper management to manage with real-time data, resulting in more informed decisions.

The future of Enterprise Information Systems will lie in the successful migration away from mainframe computer systems. This will eliminate the need to learn different computer operating systems and reduce the overall cost of implementation. This trend will also utilize existing COTS software applications and minimize the need to learn a new or special language for the EIS package. New Enterprise Information Systems will exist on personal computers and make use of Windows open systems computing and, component based object-oriented programming. These systems are easier to build and maintain and are less expensive than existing EIS. In addition, these systems provide access to organization and external information through visual screens that combine text, numerical data, graphical data and images. The visual screens will provide users with quick and easy access to data, thereby improving their decision-making capabilities.

2. Software

Software designed to directly manipulate data is an important tool in designing an effective EIS. Therefore, the software components and how they integrate the data into one system are very important.

The data supplied to an EIS can be obtained from several different sources. The most common sources are databases residing on a range of vendor-specific and open computer platforms. The primary activities regarding the database information will be inserting data, modifying current data, and querying stored data.

The structure of the database determines the method of access. Most database systems maintained by the DOD use relational design architecture. The advantages of this structure are that they can be easily expanded or updated, are simple to use, and can be accessed in several different formats. The relational database provides the flexibility that is especially valuable in the distributed or Client/Server environment. Almost all databases have extensions to allow for access via the Internet.

3. Interface

From the users perspective an EIS is the interface. In other words, all that the user will see is the Graphical User Interface (GUI) component of an EIS, which is comprised of an applet in our prototype. Therefore, when designing the interface, emphasis should be placed on usability and simplicity (i.e. the “Front-end”). The ideal interface for an EIS would have the following properties:

- Ease of use
- Be consistent with the mission of the organization.
- Provide informative and explanatory error messages
- Be highly flexible, in order to handle users at all levels of computer proficiency

The various types of interfaces available for an EIS are shown in Table 2.3.

Type	Description
Scheduled reports	Batch-oriented Predefined, prepared reports Not flexible No interaction required
Questions/answers	Interactive Ad hoc in nature
Menu-driven	User friendly Step-by-step procedures
Natural language	Regular English is used to interact with the EIS
Input/output	Predefined data/information relationships are known by the user

Table 2.3: Types of interfaces available to the EIS [Ref: 2]

4. Telecommunications

The current trend in organizations is to decentralize operations. Telecommunications will play a pivotal role in many networked information systems due to this trend. A reliable network is necessary to transmit data from one site to another. The requirement for quick access to distributed data, increases the importance of telecommunications within an EIS.

A successful Enterprise Information System should have the following traits:

- It should be easily accessible by all users in an organization
- Easy for the novice user to operate
- It should be flexible regarding hardware/software modifications
- It should be relatively inexpensive to build and maintain
- It should be PC based to cut down on costs and the system should be built to allow for quick and affordable upgrades to stay up to date with changes in technology.

An EIS within the Naval Security Group will provide for the easy and rapid dissemination of data. In addition, the Naval Security Group will greatly profit in reduced man-hours and computer resources by implementing this architecture. The following chapters will give the reader the background information required to understand and analyze our sample Enterprise Information System, which was implemented using JDBC and Common Object Request Broker Architecture (CORBA).

III. RELATIONAL DATABASES AND THE STRUCTURED QUERY LANGUAGE

A database is a logically unified collection of data that has some inherent meaning and represents some aspect of the real world. In addition, a database is designed and built to accomplish a specific requirement of an organization or individual. For example, the data that is related to the inventory of a military supply center could be stored in a database.

A Database Management System (DBMS) is comprised of programs that enable users to perform the following:

- **Define a database** – this involves specifying data types, structures, and constraints for the data to be stored in the database.
- **Construct a database** – is the process of storing the data on a storage medium controlled by the DBMS.
- **Manipulate a database** – includes functions such as querying the database, updating the database, and generating reports from the data.

A Database Management System is simply a COTS application that can easily be installed and used on all types of computer systems.

Within the realm of Database Engineering there are four basic types of databases: Relational, Network, Hierarchical and Object-oriented databases. The most popular type in use today is the Relational database model. Therefore, the focus of this chapter and the implementation of our prototype will be the Relational database model. The Relational database model is based on a simple and uniform data structure called a *relation*. Each relation is defined as a table of values. Within a table, each row is called a *tuple* and represents a unique value in the database. In addition, each table must have a unique column value called a *primary key* that is used to identify tuples in the relation. Any column that is part of a primary key cannot be null. This rule is referred to as *entity integrity*. Table 3.1 illustrates a relational database table.

DepartmentNumber	DepartmentName	ManagerName
32001	Operations	Ted Thompson
32012	Budgeting	Mark Bodine
32014	Research	Ted Thompson

Table 3.1: Relational Database

In the table above, each column represents a different attribute of a department. Each row represents a different department. The DepartmentNumber attribute is unique for each department, so it is chosen as a primary key.

Foreign keys are used to define relationships between tables. A foreign key in one table is a reference to a primary key in another table. A tuple that refers to another tuple must refer to an existing tuple in that relation. This rule is referred to as *referential integrity*. In the sample table above, the ManagerName attribute is chosen as a foreign key, because it refers to a unique row of data in the Manager table, which has this value as its primary key.

Designing a relational database involves deciding which attributes belong to a relation, choosing proper names for the columns of the table, specifying the data types and domains of these attributes, choosing primary keys and specifying relations between the tables by the use of foreign keys.

SQL is a comprehensive database language designed for use with relational databases. It has statements for data definition, query and update. SQL consists of a set of standard commands that can be understood by all compliant Relational Database Management Systems (RDBMS). The following is a list of more commonly used SQL commands.

1. Data Definition Language:

The following commands are used to create or modify tables and other database objects and are used extensively in our prototype.

- **CREATE TABLE** - this command is used to create a table with the name of the table and columns provided by the user. The following is the command used to define a table with the name Department:

CREATE TABLE Department

```
(DepartmentName      VARCHAR(10)          NOT NULL,
 ManagerName          CHAR (15)            NOT NULL,
 DepartmentNumber     INT                  NOT NULL,
 PRIMARY KEY (DepartmentNumber),
 FOREIGN KEY (ManagerName) REFERENCES Manager (Name));
```

- **DROP TABLE** - this command is used to delete a table definition and all rows in the table. The following is the command used to delete the table defined in the previous example:

DROP TABLE Department;

- **ALTER TABLE** – the definition of a table can be changed by using this command. It is possible to add or drop a column, change a column definition, or add/remove constraints defined for the table. The following example shows the command used to add another column to the Department table defined above.

ALTER TABLE Department ADD ManagerStartDate DATE;

2. Data Manipulation Language:

The following commands are used to insert, delete, query and display data from a RDBMS. These commands are used extensively in our prototype.

- **SELECT** – This is the most commonly used SQL command and is used to query the database and display selected data to the user. We have used SELECT queries to retrieve data from the database that met the search criteria in our EIS prototype. The name of the table from which the data is to be extracted from and the names of the columns should be specified in the query. The following is a sample SELECT query.

```
SELECT DepartmentName, DepartmentNumber
FROM Department
WHERE ManagerName = 'John Lewis';
```

- **INSERT** – INSERT commands are used to add new rows to a table. It can be used to fill a new table with data or add new data to an already existing table. When the user of the EIS performs an operation to input new data in to the database the request is carried out by means of an INSERT query. The example below adds new department to the table defined above.

INSERT INTO

```
Department (DepartmentName, ManagerName, DepartmentNumber)
VALUES ("Research", "John Lewis", 32015);
```

- **DELETE** – This command removes a single row or rows of data that meet the condition specified. The following query can be used to delete the department whose manager is "John Lewis".

```
DELETE FROM Department
WHERE ManagerName = 'John Lewis';
```

- **UPDATE** – This command is used to modify a single column or columns of data in the row that meets the condition specified. UPDATE queries are used in the EIS to modify existing data in the back-end database. The following

query can be used for example to change the manager name of the department that has a DepartmentNumber of 32014.

```
UPDATE Department
SET ManagerName = 'Johnson'
WHERE DepartmentNumber = 32014;
```

There are a variety of ways to use SQL to define and manipulate data in a Database Management System. One way is to embed SQL statements in a high-level programming language. This is called Embedded SQL. Embedded SQL allows programmers to place SQL statements into the host language. SQL statements are delimited with specific starting and ending statements defined by the language. When compiling a program with embedded SQL statements, a precompiler translates these statements into equivalent host language source code. After precompiling, the host language compiler compiles the resulting source code. The SQL statements, that were extracted, form a database module that is parsed, validated and executed by the DBMS.

Another way to execute SQL statements is to have pre-defined and compiled procedures, which reside on the database and can be called by clients. These procedures are commonly referred to as stored procedures. Stored procedures offer many benefits when the query to be executed is large and complex. The following is a list of these benefits:

- After a stored procedure is executed for the first time, it does not need to be parsed, optimized or compiled again.
- Stored procedures can consist of multiple queries, but can be executed with a single statement, thus reducing network traffic.
- Stored procedures accept input parameters so that multiple clients using different input data can invoke a single stored procedure.
- Stored procedures are much faster in Client/Server systems, which will be explained, in Chapter IV.

The problem with stored procedures is that they are vendor-specific, totally non-standard, not portable across platforms, and have no standard interface definition language or stub compiler. Therefore, there is no standard way to pass or define parameters.

Another alternative to Embedded SQL is to use a callable SQL Application Programming Interface (API) for database access. An API does not require a precompiler to convert SQL statements into a high-level language, which can then be compiled and executed on the database. Instead, an API allows the user to create and execute SQL statements at run time. A standard API can be used to produce portable applications that are independent of any database product. The SQL Access Group Call Level Interface (SAG CLI) specifies a common API for accessing multiple databases. It provides common SQL semantics and syntax, codifies the SQL data types, and provides common error handling and reporting [Ref: 3]. The SAG API enables the client to connect to a database, execute requests, retrieve the results and terminate the connection. Table 3.2 provides a comparison of the features of CLI and Embedded SQL.

Features	X/Open SQL Call-Level Interface	ISO SQL-92 Embedded SQL(ESQL)
Requires target database to be known ahead of time	No	Yes
Supports Static SQL	No	Yes
Supports dynamic SQL	Yes	Yes
Supports stored procedures	No	No
Applications must be precompiled and bound to database server	No	Yes
Easy to program and debug	No	Yes
Tool friendly	Yes	No
Easy to package	Yes	No

Table 3.2: Comparison of CLI and Embedded SQL features [Ref: 4]

Microsoft's ODBC is a Windows API that is an extended version of the SAG CLI. In addition to its basic functionality, it provides methods to retrieve information about the database and handle multimedia types of data. ODBC offers the ability to

connect to multiple kinds of databases on different platforms. However, the following are its drawbacks:

- It is procedure oriented and thus does not mold with most of the application programs written in an object-oriented language.
- ODBC standards are controlled by one vendor and are subject to change at the vendor's wish.
- ODBC is hard to learn and debug. It mixes simple and advanced features together.
- ODBC driver manager and drivers must be installed on every client machine. This means it would be a poor choice for a web-based database system.
- It has drawbacks in the security, robustness and portability of applications [Ref: 5].

Because of these drawbacks, and since Java is the natural language of choice for an Internet based database system, JDBC was developed by Sun Microsystems as a high-level API for invoking SQL commands directly on different vendor databases. JDBC provides the security, robustness and portability that ODBC lacks. JDBC is a Java API that enables large-scale applications to provide "pure Java" solutions. We will introduce and explain the JDBC API in Chapter V.

Three standard types of database architecture models have been developed based on the physical distribution of components in a DBMS. In the first model all components that enable the user to operate and maintain the data are located on a single machine. This is typically referred to as a "Stand-Alone" DBMS. The drawback of this approach is that it doesn't allow sharing of the database between multiple users. The first generation of Enterprise Information Systems were maintained on a single mainframe computer utilizing this model.

Advances in computer communications and networking led to the second generation of Database Architecture Models, which had the ability to integrate different flavors of locally maintained databases into one large distributed DBMS. In this system a search request generated by a user is distributed to multiple databases and the results are

returned more rapidly. The drawbacks of these systems are the maintenance overhead, cost and performance. [Ref: 1]

The latest generation of Database Architecture Models is referred to as the Client/Server model. It comprises the positive aspects of both the stand-alone and Distributed Database Architecture Models. In a Client/Server architecture model the data processing is divided between the client and the database server, which are connected by a communications network. This model is the focus of our thesis and will be further discussed in Chapter IV.

IV. CLIENT/SERVER ARCHITECTURE

The term Client/Server is an application architecture that enables a computerized application to be broken up into two or more less complex tasks with a communication mechanism for these sub-processes to cooperate [Ref: 5].

Typical examples of application layers are:

- **Presentation Logic:** Handling how the user interacts with the application. Usually implemented by providing an easy to use Graphical User Interface (GUI).
- **Business Logic:** Handling the business rules of the application.
- **Data Access Logic:** Handling the storage and retrieval of data.

The forms of Client/Server systems in use today are one-tiered, two-tiered, three-tiered and N-tiered architectures. All of these Client/Server systems have the following distinguishable properties [Ref. 6]:

- **Service:** Client/Server is primarily a relationship between processes running on separate machines. The server process is a provider of services. The client is a consumer of services. In essence, Client/Server provides a clean separation of function based on the idea of service.
- **Shared resources:** A server can service multiple clients at the same time and regulate their access to shared resources.
- **Asymmetrical protocols:** There is a many-to-one relationship between clients and server. Clients always initiate the dialog by requesting a service. Servers are passively awaiting requests from the clients.
- **Transparency of location:** The server is a process that can reside on the same machine as the client or on a different machine across a network. Client/Server software usually masks the location of the server from the clients by redirecting the service calls when required. A program can be a client, a server, or both.

- ***Mix-and-match***: The ideal Client/Server software is independent of hardware or operating system software platforms. You should be able to mix-and-match client and server platforms.
- ***Message-based exchanges***: Clients and servers are loosely coupled systems that interact through a message-passing mechanism. The message is the delivery mechanism for a service request and reply.
- ***Encapsulation of services***: The server is a “specialist.” A message tells a server what service is requested; it is then up to the server to determine how to get the job done. Servers can be upgraded without affecting the clients as long as the published message interface is not changed.
- ***Scalability***: Client/Server systems can be scaled horizontally or vertically. Horizontal scaling entails adding or removing client workstations with only a slight performance impact. Vertical scaling entails migrating to a larger and faster server machine or multiservers.
- ***Integrity***: Server code and server data is centrally maintained, which results in cheaper maintenance and the guarding of shared data integrity. At the same time, the client remains personal and independent.

The Client/Server properties identified above describe how data can easily be accessed across a network. They also provide a template for Client/Server design architecture.

The creation of Enterprise Information Systems grew from the problems associated with monolithic mainframe systems. Each of these systems contained their own presentation object, business logic and access mechanisms. They could not share data with other systems, so each had to store a private local copy. This resulted in redundant copies in an organization. The inefficiency and cost of these systems led to Relational database technology and Client/Server systems that use more than one tier. This was made possible by new technologies like networks, low-cost personal computers and RDBMSs.

First generation Client/Server systems were created using a two-tiered architecture, where a client presents a GUI to the user, and utilizes the user's data entry and actions to perform requests of a database server running on a different machine. Application logic is tied to the client application and a network process is required to mediate the Client/Server interaction. These systems can be implemented quickly using rapid application development tools.

The problems associated with placing the application logic with the client are the requirement for processing power and fast network connections to handle large result sets that may be returned in response to database requests. If the logic changes, the effort in updating software on numerous workstations can be excessive and lead to high costs in maintenance and support.

An alternative implementation is to colocate the application logic on the host with the database using stored procedures. This does not solve all of the problems. The need for the database to maintain a separate session for each client will result in the rapid consumption of back-end server resources.

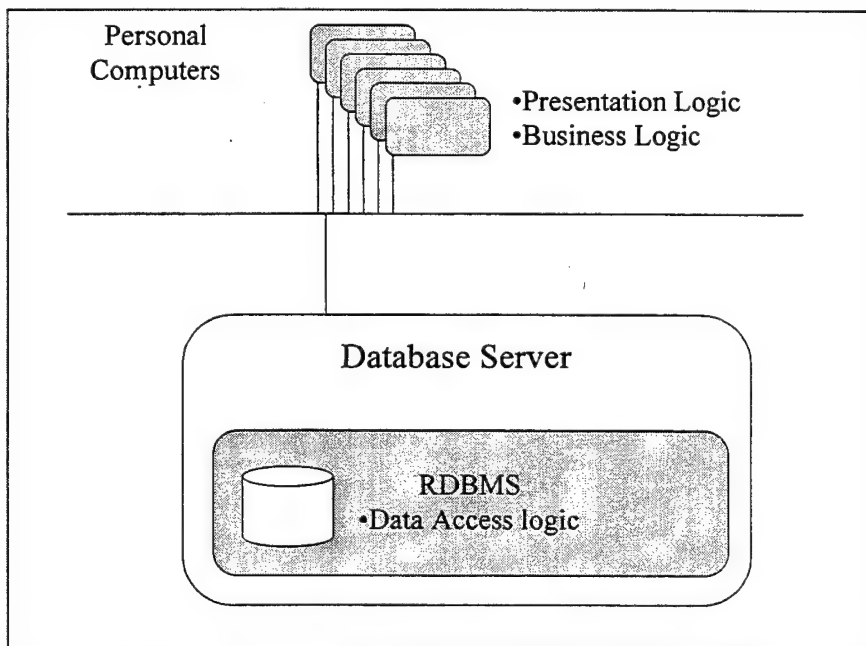


Figure 4.1: Two-Tiered Client/Server model [Ref: 5]

Both of these implementations reduce the possibility for flexibility and portability due to the proprietary nature of the underlying models. In general, two-tiered systems do

not provide the scalability for large-scale applications deployed across a network or the Internet. Figure 4.1 presents the two-tier Client/Server model.

The three-tier Client/Server architecture extends the basic two-tier Client/Server model by adding a middle tier to support the application logic and common services. The client interacts with the middle tier via a standard protocol such as TCP/IP. The middle tier interacts with the server via appropriate database protocols. In addition, the middle tier contains most of the application logic, translating client calls into database queries and translating the results returned from the database into a client viewable format. Figure 4.2 describes the three-tier Client/Server model.

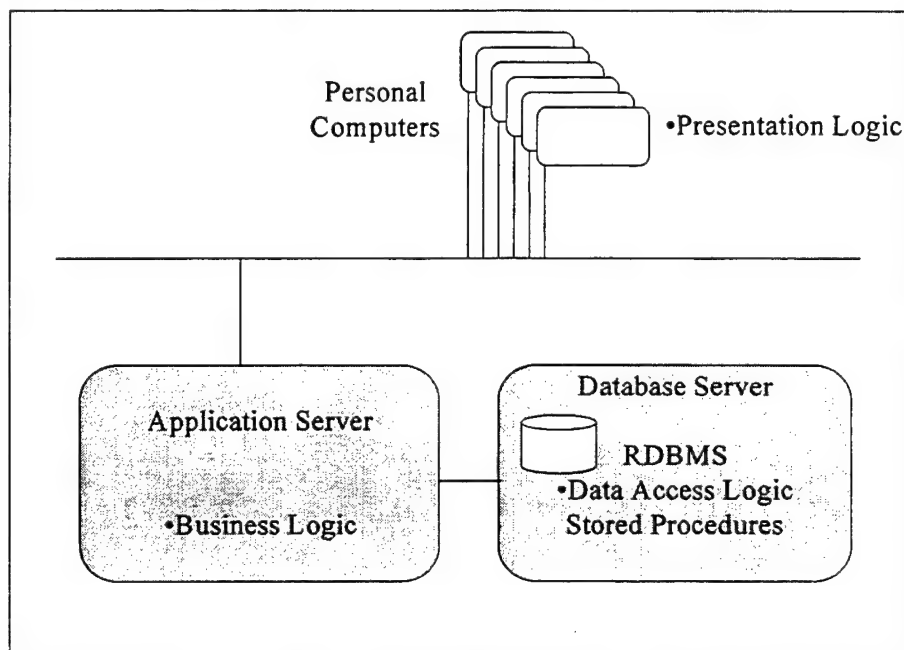


Figure 4.2: Three-Tier Client/Server model [Ref: 5]

By creating three tiers, the application can be partitioned into the presentation logic, business logic and data access logic. The advantage is that any of the tiers can be enhanced or replaced without affecting other tiers. This approach does not require a separate database connection for each user, instead many user sessions can be funneled into a few database connections enabling savings of system resources, such as processing and memory on the database server. While two-tier systems remain suited for simple applications, three-tier Client/Server solutions are recognized as the ideal choice, since

they are more maintainable and supportable, and are flexible to adapt to ever-changing requirements.

The three-tier architecture can be extended to N-tiers when the middle tier provides connections to various types of services, integrating and coupling them to the client, and to each other. An N-tiered system can also be created by partitioning the application logic among various hosts. As requirements change, the partitioning and deployment of the system can be reviewed and modified with minimal impact. CORBA gives you the capability to implement an N-tiered architecture due to its flexibility. This implementation remains largely theoretical in nature, as most organizations operating in a Client/Server environment are currently utilizing a three-tier architecture. Figure 4.3 describes the N-tier Client/Server model.

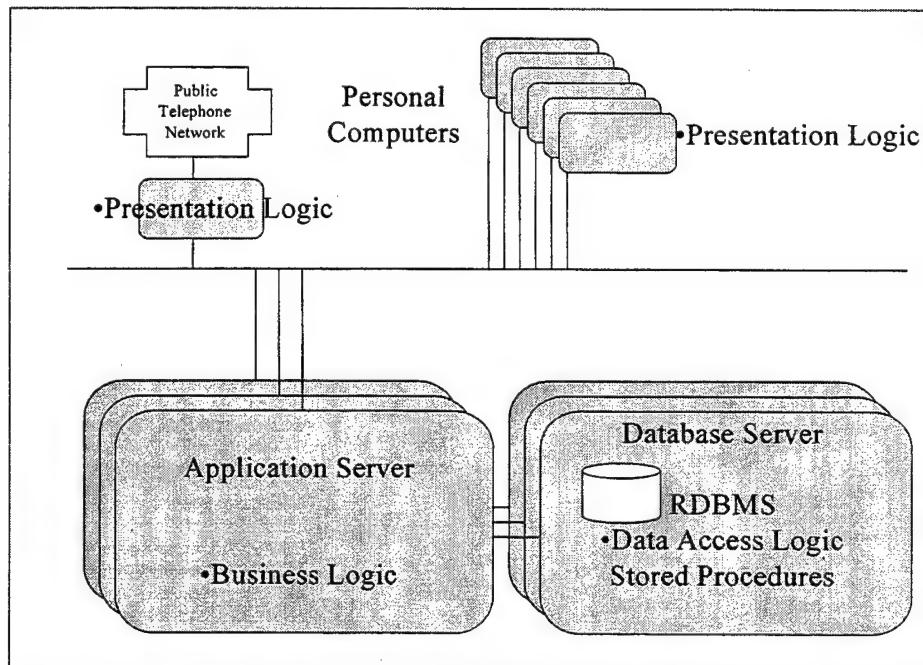


Figure 4.3: N-tier Client/Server model [Ref: 5]

A. COMPONENT BASED THREE-TIER CLIENT/SERVER SYSTEMS

A component is an application independent object that represents a real world entity. Components can be used in multiple applications without any modification. Components contain a user interface, maintain their state and can cooperate with other components. Component-based architectures are considered the ideal technology for

three-tier Client/Server systems. The main reason is due to their ease of expandability. In our EIS prototype, the components that make up the first tier present a view of the application to the user. The middle tier components enforce the business logic. The server components have access to the back-end information source and contain the data access logic. Any of the components in the system can be modified or replaced without redesigning the entire application. Clients interact only with the front-end components. Front-end components communicate with middle tier components, which in turn interact with back-end components to execute any operation requested by the client. We will cover all of these components beginning with the back-end components, which are responsible for accessing the database. These components use JDBC to insert, modify and retrieve data from the database server. In order to understand these components, the reader needs to be familiar with Java and the JDBC package in particular. Chapter V will cover the essential information regarding Java and JDBC and how they can be used to implement a Client/Server system.

V. JAVA AND JDBC

A. JAVA

Java is an object-oriented programming language that can be used to create applets, which are programs that can be embedded in a web page. Instead of web pages with text and static graphics, Java applets can make use of audio, animation, interactivity and video imaging.

One of the biggest advantages of Java is that it is portable. Therefore, an application written in Java is platform independent. Any computer with a Java-capable browser can run Java applets. Developers don't need to modify applets or stand-alone applications code when changing platforms. Additionally, most organizations retain existing legacy code that they do not wish to convert to Java. Because Java was created with the capability to interface to existing C and C++ code, this provides a workable solution to a problem most organizations face.

With other programming languages the compiler creates platform specific machine language code. In comparison, the Java compiler creates Java byte-code, which in turn is interpreted by the Java Virtual Machine (JVM) at run-time. Therefore, an application can be run on any platform for which a Java Virtual Machine Implementation exists.

The Java programming language was designed from the start as an object-oriented programming language. Object-oriented languages enable designers to break up large projects into easily manageable components. These components can be modified and re-used with considerably less effort than monolithic applications.

Java is a natural language for writing applications for distributed systems because of the following:

- **Garbage collection:** with other programming languages a significant burden on the programmer is the allocation and de-allocation of memory. Memory leaks in a program can cause an application to crash. In Java, if an object is no longer being used, it is automatically removed from memory by the Java

garbage collector. C++ for instance requires that the programmer keep track of the allocation and deallocation of memory, which is not a concern in Java.

- **No Pointers:** this feature removes a significant source of errors in computer programming. Java utilizes “object references” instead of memory pointers. This eliminates problems concerning pointer arithmetic and “out of bounds” memory access errors.
- **Multi-threaded:** the ability of a program to do more than one thing at a time. This is important in multi-media and network applications, due to the CPU intensive nature of these applications and the inherently slow speed of network connections.
- **Strong Typing:** Java enforces strong type checking, therefore many errors are caught at compile time. This significantly cuts down on run-time errors.
- **Scalability:** the Java platform is designed to scale well. The small footprint of the JVM and its optimized byte-code enable Java-based applications to run on a wide range of platforms, from smart-cards with limited processing capabilities to mainframes.
- **Security:** the JVM is designed with security in mind and automatically subjects all programs to byte-code verification. The byte-code verifier checks that the format of incoming code is correct, meaning that it doesn’t forge pointers, doesn’t violate access restrictions and uses objects only for what they were intended.

Java programs go through five phases in order to be executed. These are edit, compile, load, bytecode verify, and execute. Figure 5.1 describes the specifics of a Java development environment.

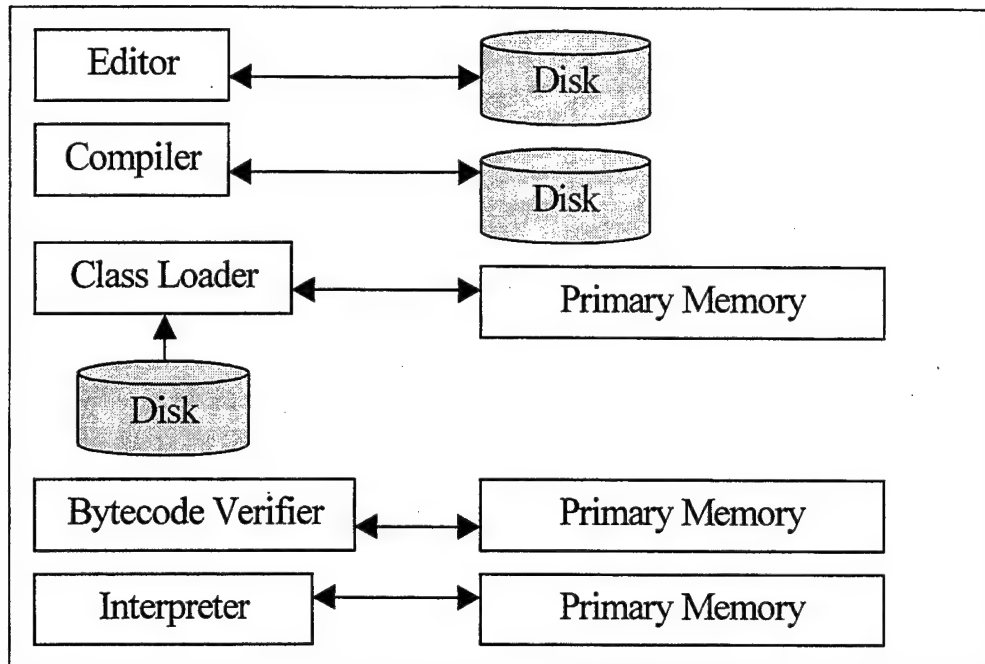


Figure 5.1: Typical Java environment [Ref: 8]

- **Editor** – the program is created and stored on disk.
- **Compiler** – creates the bytecode and stores it on disk.
- **Class Loader** – places bytecode into memory.
- **Bytecode Verifier** – confirms that all bytecode is valid and doesn't violate Java's security restrictions.
- **Interpreter** – reads bytecode and translates it into a language that the computer can understand, possibly storing data values as the program executes.

B. JDBC

JDBC is a high level Application Programming Interface (API). Developers use the JDBC API classes and interfaces to make necessary transactions with a RDBMS, while JDBC driver vendors use JDBC to model their drivers accordingly. JDBC consists of a set of classes and interfaces written in the Java programming language and provides a standard API for database developers, making it possible to write database applications using a pure Java API.

With the JDBC API, programs written to access a database do not require extensive modification when the backend database has been changed. A program written with the JDBC API can send SQL statements to multiple database architectures simultaneously. This enables organizations to continue to use legacy database management systems, as well as, modern Client/Server systems.

Development time for new applications is short and installation and version controls are greatly simplified. A programmer can write an application or an update once, place it on the server, and everybody has access to the latest version.

JDBC makes it possible to easily perform the following tasks:

- Establish a connection with a database.
- Send SQL statements.
- Process the results.

The following describes each JDBC capability in detail:

1. Establishing a connection with the database:

This involves two steps: loading a JDBC driver and making a connection. Loading the driver is accomplished by asking for an instance of the driver explicitly, as in the following line:

```
Class.forName ("sun.jdbc.odbc.JdbcOdbcDriver");
```

A connection is established by using the appropriate driver to connect to the DBMS. The corresponding sub-protocol identifier is appended after the keyword jdbc:. The following code explains how this is accomplished:

```
String url="jdbc: ff-microsoft: //cryptologist.cs.nps.navy.mil" ;  
String login = "sa";  
String password = "test";  
  
Connection con = DriverManager.getConnection
```

```
(url, login, password);
```

2. Sending SQL statements:

In order to execute a SQL statement on a relational database using JDBC, a statement object must be created. The SQL statement to be executed is supplied as an argument to the proper method of the statement object. The type of method differs according to the query being executed. The following code shows how to create a statement object and execute a simple UPDATE query:

```
String query = "UPDATE SAILORS " +  
               "SET PHONE_NUMBER = 555-2111 " +  
               "WHERE LAST_NAME = Lewis ";  
Statement stmt = con.createStatement ();  
stmt.executeUpdate(query);
```

3. Processing the results:

When a SELECT statement is executed, the results of the query will be returned in a ResultSet object defined in the JDBC package. The tuples in the ResultSet can be retrieved using the next() method defined in the ResultSet class. The following code shows how to create and execute a SELECT statement, retrieve the results and display the tuples in the ResultSet object to the screen.

```
String query = "SELECT COMMAND_NAME FROM " +  
               "COMMAND_INFORMATION " +  
               "WHERE COMMAND_ID = 93162";  
ResultSet rs = stmt.executeQuery(query);  
While (rs.next()) {  
    String s = rs.getString("COMMAND_NAME");  
    System.out.println ("The name of the command is: " + s);  
}
```

C. JDBC INTERFACES

The JDBC API consists of six classes and eight interfaces. A complete representation of these classes and interfaces is shown in Figure 5.2. Arrows denote inheritance.

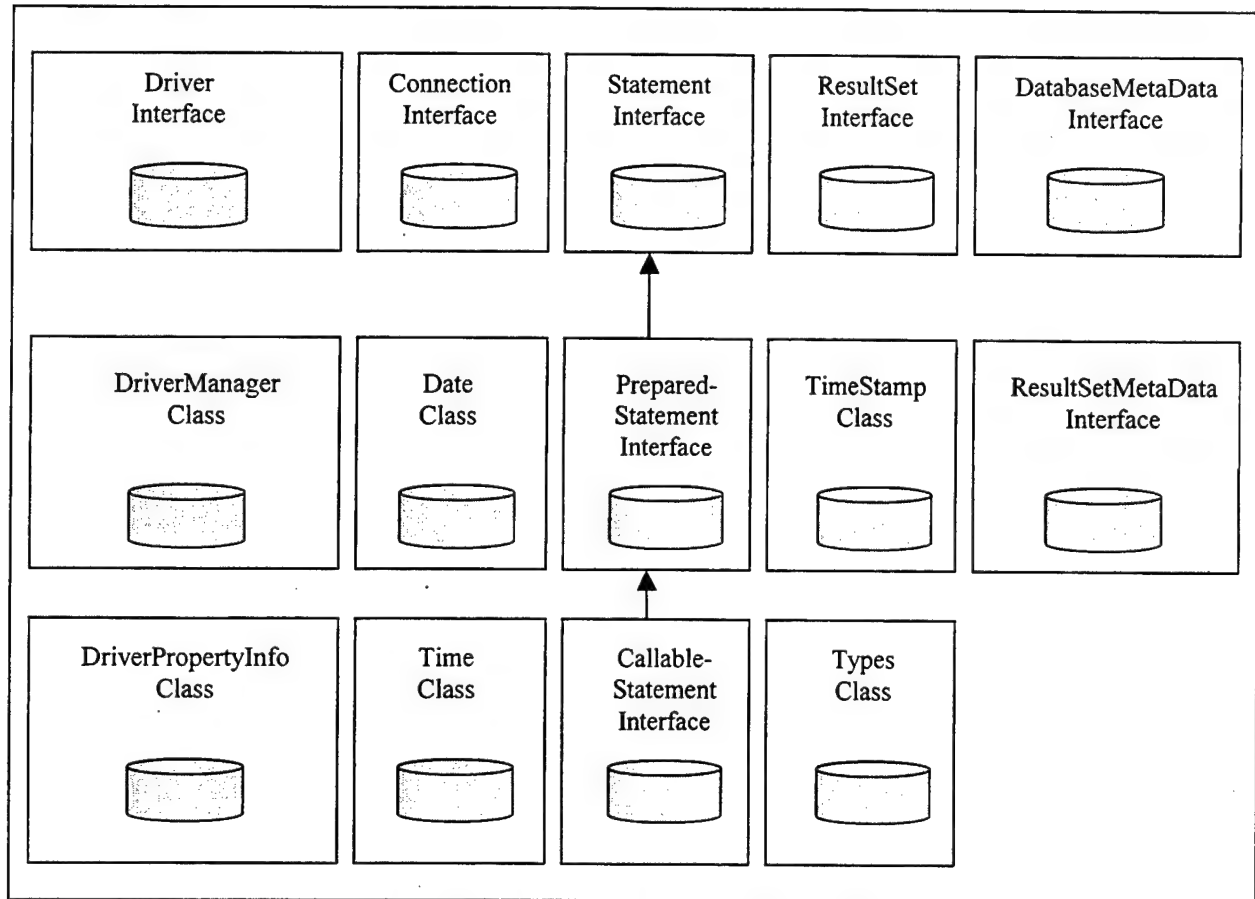


Figure 5.2: Diagram of all JDBC interfaces and classes [Ref: 9]

These classes and interfaces represent database connections, SQL statements, result sets and database metadata. The most important classes and interfaces will be discussed in the following sections.

1. DriverManager:

The DriverManager keeps track of available drivers and handles the creation of connections between databases and appropriate drivers. By invoking the `getConnection()` method on this interface, a valid connection with the database can be established.

2. Connection Interface:

A connection represents a session with a specific database. Within the context of a connection, SQL statements are executed and results are returned [Ref: 10]. An application can have one or more connections with a single database or it can have simultaneous connections with multiple databases. The established connection passes SQL statements to the connected database.

3. Statement Interface:

SQL statements without parameters are normally executed using this object. The Statement object can be created by calling the `createStatement()` method of a connection object. The `execute` method can be invoked to execute SQL statements:

```
boolean execute(String arg) throws SQLException;
```

This method executes `arg`, which is a SQL statement that may return one or more result sets, one or more update counts, or any combination of these. This method is useful if the designer doesn't know whether the statement will be an update or a query operation. A call to this method executes a SQL statement and returns `true` if the result is a `ResultSet` and returns `false` if the result is an update count.

```
ResultSet executeQuery(String arg) throws SQLException;
```

This method executes `arg`, which is a SQL statement that returns a single result set representing the results of the provided query.

```
int executeUpdate(String arg) throws SQLException;
```

This method executes a SQL INSERT, UPDATE, or DELETE statement that doesn't have parameter placeholders. It may also be used to execute SQL statements which return no value, such as CREATE TABLE or DROP TABLE.

4. PreparedStatement Interface:

If the same SQL statement is executed many times, it is more efficient to use a PreparedStatement. A SQL statement with or without IN parameters can be pre-compiled and stored in a PreparedStatement object. A PreparedStatement object can be more efficient than a Statement object, because it has been pre-compiled and stored by the database. In order to benefit from pre-compilation, the JDBC driver must support this feature. The PreparedStatement object can be created by calling the prepareStatement() method of a connection object.

5. CallableStatement Interface:

A callableStatement is used to execute SQL stored procedures. These stored procedures are SQL statements that can be invoked by a name. Various DatabaseMetaData methods can be used to retrieve information regarding which stored procedures a DBMS supports and the descriptions and names of these procedures. CallableStatement objects are created with the connection method prepareCall ().

6. ResultSet Interface:

A ResultSet object provides access to a table of data generated by executing a SQL statement. Table rows are retrieved in sequence. Within a row column values can be accessed in any order [Ref: 4]. A ResultSet maintains a cursor pointing to its current row of data. In order to retrieve the first line of data, the next() method must be invoked. Various getXxx() methods can be invoked to retrieve different column values. The SQL data types, the corresponding Java data types, and the recommended method calls for conversion are shown in Table 5.1.

SQL Data Type	JAVA data type	Recommended getXXX Conversion method
CHAR	String	getString()
VARCHAR	String	getString()
LONGVARCHAR	String	getAsciiStream() getUnicodeStream()
NUMERIC	java.math.BigDecimal	getBigDecimal()
DECIMAL	java.math.BigDecimal	getBigDecimal()
BIT	boolean	getBoolean()
TINYINT	byte	getByte()
SMALLINT	short	getShort()
INTEGER	int	getInt()
BIGINT	long	getLong()
REAL	float	getFloat()
DOUBLE	double	getDouble()
FLOAT	double	getDouble()
BINARY	byte[]	getBytes()
VARBINARY	byte[]	getBytes()
LONGVARBINARY	byte[]	getBinaryStream()
DATE	java.sql.Date	getDate()
TIME	java.sql.Time	getTime()
TIMESTAMP	java.sql.Timestamp	getTimestamp()

Table 5.1: SQL and Java data types and recommended conversion methods

D. JDBC AND CLIENT/SERVER MODELS

The JDBC API supports database access, utilizing two-tier and three-tier models. In the two-tier model, a Java applet or application communicates with the database. This requires a JDBC driver that can communicate with the particular DBMS being accessed. A user's SQL statement is delivered to the database and the results of the statement are returned to the user. The database may be located on remote machine to which the user is connected via a network. This is referred to as a Client/Server configuration, with the user's machine as the client, and the machine hosting the database as the server. The network in question can be an Intranet or an Internet. Since all code is executed on the client, this implementation may not scale very well. An application that requires multiple concurrent client connections to a database will likely be a huge burden for both the network and the DBMS. A two-tier driver may also be unable to take advantage of the multi-threading capabilities of Java, if the client libraries themselves are not multi-

threaded. Threads are an important means for increasing performance because they allow multiple concurrent transactions on database data. Java has built-in multi-threading support, as well as ways to synchronize multi-threaded operations, but if the client libraries are not thread-safe, the JDBC driver cannot take advantage of this capability.

Most Java virtual machines restrict applets from connecting to any machines other than the host that served the applet. The applet therefore cannot connect directly to any local or third-machine databases. Because of this restriction in a two-tier architecture, the applet realizing the user interface must be stored on the same server where the database is stored. Figure 5.3 describes the two-tier model using JDBC.

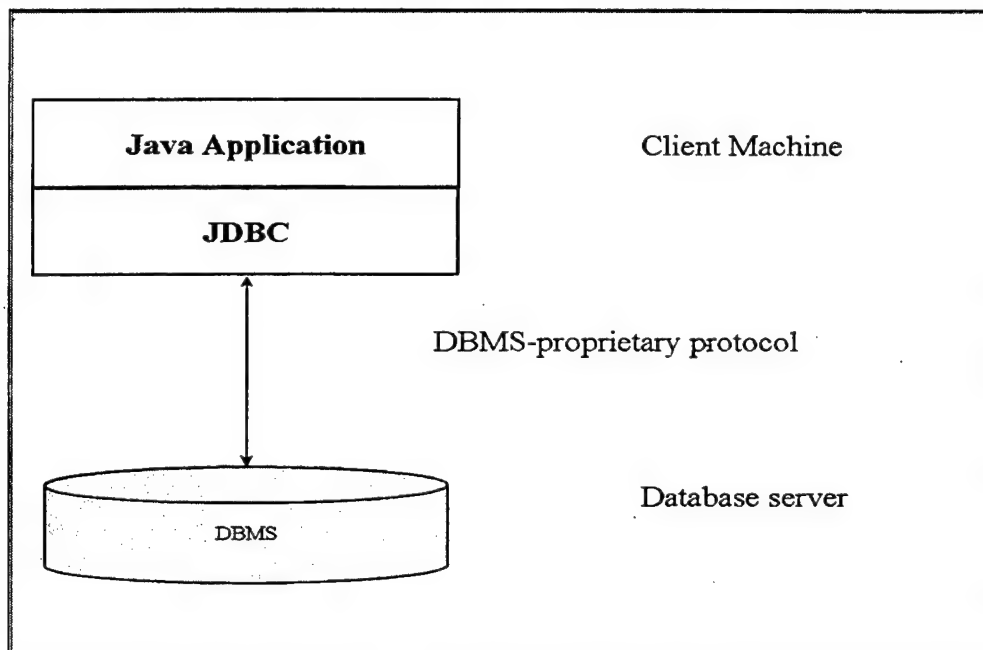


Figure 5.3: JDBC Two-Tier model [Ref: 8]

In the three-tier model, commands are sent to a middle tier of services, which then send SQL statements to the database. The database processes the SQL statements and sends the results back to the middle tier, which then passes them to the user. This makes the three-tier model very attractive, because the middle tier makes it possible to maintain control over access and the type of updates that can be made to archived data. Another advantage is that when there is a middle tier, the user can employ an easy-to-use higher-level API, which is translated by the middle tier into appropriate low-level calls. The

client can operate as a multi-threaded application and let the intermediate server handle the synchronization. In a three-tier architecture the middle layer can hide information about the database server from the applet. Only the middle tier knows how to find and manipulate the data. The secure intermediate server can provide the means to shield the client from direct access to DBMS by providing a username and password because the identification and authentication is accomplished at the server level. Figure 5.4 describes the three-tier model using JDBC.

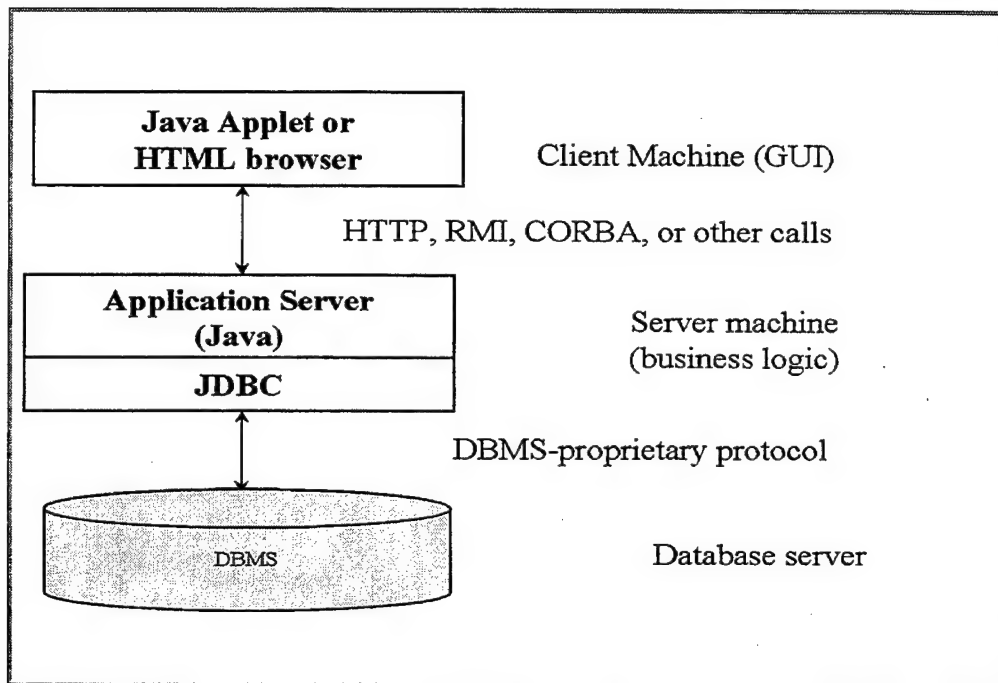


Figure 5.4: JDBC Three-Tier model [Ref: 8]

E. JDBC DRIVERS

Drivers are Java classes that implement the driver interface. The DriverManager will load and use an instance of the driver class to answer connection requests. When a driver class is loaded, it should create an instance of itself and register with the DriverManager. The driver interface provides six methods, which are used to set up a connection between a driver and a database, provide information about the driver, or get the information necessary for making a connection to a database. [Ref: 9]

There are various choices for driver implementations. Figure 5.5 depicts the JDBC driver implementation. The following are the various classes of JDBC drivers available:

1. JDBC-ODBC bridge plus ODBC Driver:

A JDBC-ODBC bridge implemented in JDK 1.1 provides JDBC access to databases via most ODBC drivers. To use this bridge, ODBC binary code, and in some cases database client code, must be loaded on each client machine. This kind of driver is most suitable for a corporate network where client installations are not a major problem, or for application server code written in Java for a three-tier Client/Server implementation.

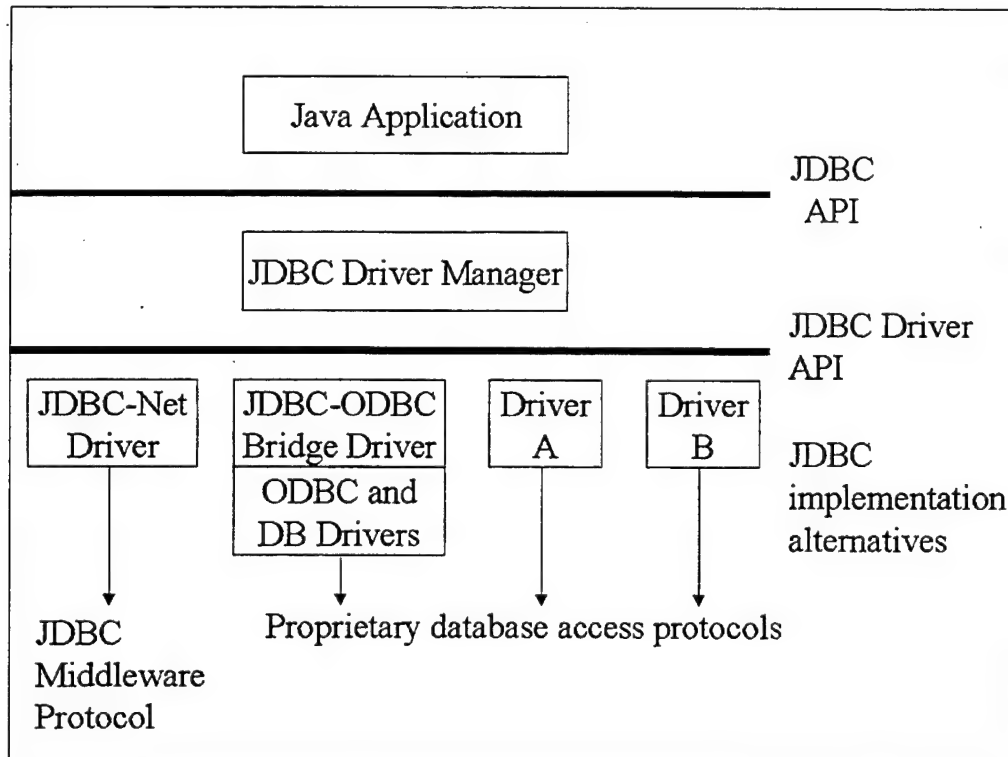


Figure 5.5: JDBC driver implementation [Ref: 8]

2. Native-API partly-Java Driver:

This type of driver converts JDBC calls on the client API to a vendor specific query language and communication protocol for use on a DBMS. Like the bridge driver, some binary code must be loaded on each client machine.

3. JDBC-Net pure Java Driver:

This driver translates a JDBC call into a DBMS independent net protocol, which is then translated to a DBMS protocol by a server. This net server middleware is able to connect all Java clients to multiple databases. The specific protocol used depends on the vendor. This type of implementation simplifies administration by loading all DBMS drivers on the middle tier application server. Of all driver implementations, this is the most flexible.

4. Native-protocol pure Java Driver:

This type of driver converts JDBC calls into the network protocol used by the DBMS's directly. This allows a direct call from the client machine to the DBMS server and is a practical solution for Internet access. Since many of these protocols are proprietary in nature, the driver can only interface with vendor specific databases. The most significant advantage of this driver is its speed, where the biggest disadvantage is the loss of flexibility.

F. JDBC SUMMARY

JDBC is a high-level interface that is used to execute SQL commands. It is much easier to use than other database connectivity API's. The primary problem with using JDBC in a web-based database environment is the requirement to install JDBC drivers on each client machine or download the driver for every connection. With slow network connection speeds, clients may be required to wait while the driver downloads. Thus, there would be a significant decrease in system performance.

In our implementation we are using JDBC on the middle tier application server, which listens for CORBA requests and then uses a native-protocol pure Java driver to execute the corresponding JDBC methods on the back-end database. This is the three-

tier JDBC model. ResultSets are parsed and packaged as CORBA data structures to be returned to the client. We will give an overview of CORBA architecture in Chapter VI and provide a detailed explanation of our EIS implementation in Chapter VII.

VI. COMMON OBJECT REQUEST BROKER ARCHITECTURE

CORBA is an industry-wide standard for creating distributed object systems. The Object Management Group (OMG) defines CORBA. OMG is a non-profit consortium of over 800 companies. The goal of this group is to provide definitions of standards for interoperable software components. CORBA specifies how software components distributed over a network can work together to perform a task without regard to the operating systems and programming languages used. OMG only facilitates the definitions of these standards; it does not deal with implementation issues. Software that is produced in accordance with these standards will be interoperable with other software that follows the same standards. CORBA also specifies an extensive set of services for creating and deleting objects, accessing them by name, storing them in persistent stores, externalizing their states, and defining ad-hoc relationships between them. [Ref: 3]

CORBA is the latest evolution of Client/Server computing. Typical Client/Server systems partition the functionality of the application, such that the client applications perform both the business processing and user interface. The server is used as the back end data source for the system, usually as a file server or a database server. Software maintenance in such a system is a burden, because the client application needs to be updated with every change of the application or business logic. Code reuse is not easy to achieve, because the rules of business are intermixed with presentation and data storage logic. These problems have resulted in the separation of Client/Server systems into three specific areas: presentation, business logic and data storage and retrieval. The tier that encapsulates the business and application logic is known as the middle tier. Implementations of the middle tier have become known as the middleware [Ref: 3]. CORBA is superior to other middleware solutions because of the following benefits it offers:

- **Open standards:** CORBA is based on open, published specifications. It is implemented by different vendors on different hardware platforms and operating systems using various programming languages. This gives the user maximum flexibility in choosing and upgrading a Client/Server system.

- ***Interoperability:*** CORBA objects are fully interoperable even when they are developed by different vendors who have no previous knowledge of each other's objects because they communicate using a common protocol, the Internet Inter-ORB protocol.
- ***Modularity:*** Every CORBA-compliant object has a well-defined interface. CORBA objects communicate with each other using these interfaces. Changing the implementation of an object does not require changes in other objects as long as the interface of that object remains the same.
- ***Coexistence with legacy systems:*** CORBA enables organizations to protect their investment. A legacy application can be encapsulated in a CORBA wrapper that defines an interface to the legacy code. This interface makes the application interoperable with other objects in the distributed environment.
- ***Portability:*** A CORBA object written on one platform can be deployed on any other platform that supports CORBA.
- ***Security:*** CORBA provides security features such as encryption, identification and authentication of the entities in the distributed system and controls access to objects and their published services.

A. OBJECT MANAGEMENT ARCHITECTURE

The OMG published the Object Management Architecture Guide (OMA Guide) in 1990. It was revised in 1992 and 1995. The OMA Guide is the highest level specification that covers all constituents of the Common Object Request Broker Architecture. The five parts of the architecture are provided in the Figure 6.1.

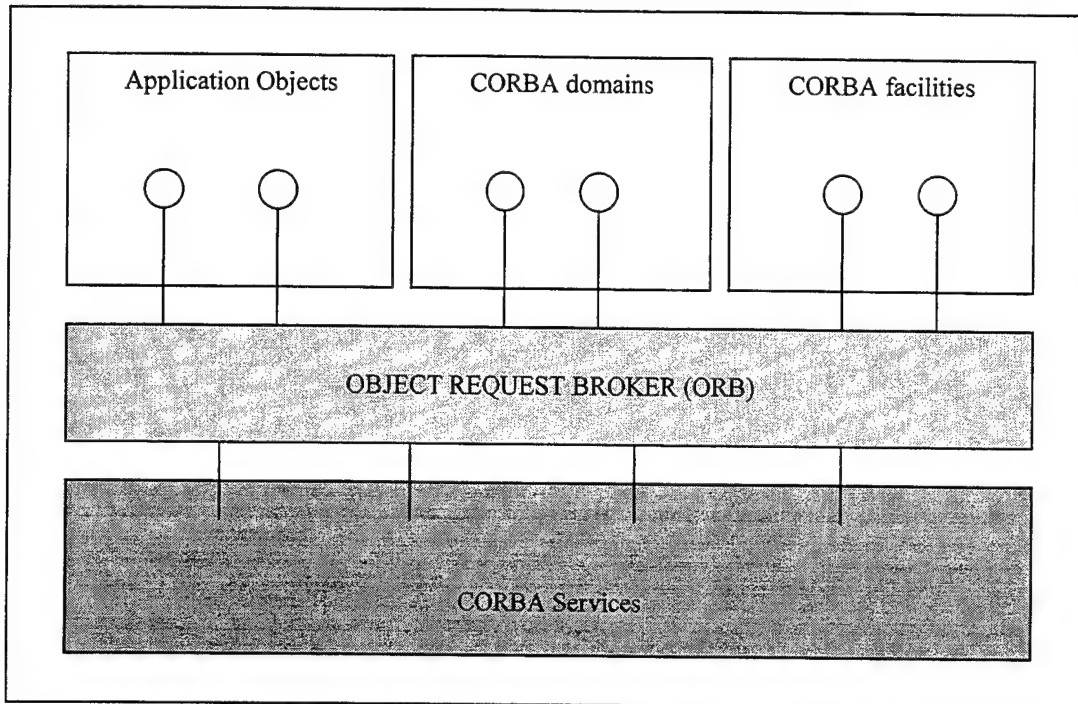


Figure 6.1: Object management architecture [Ref: 10]

a. Object Request Broker(ORB):

A CORBA Object Request Broker is the middleware that handles interactions between objects. A client machine can invoke a method on a server object that can be on the same machine or across a network using the ORB. The ORB intercepts this call and locates the object that is offering the services, providing supplied parameters to methods and returning results to the caller. The calling object does not need to know the server object's location, the programming language the server was written in or the operating system it is running on. The ORB separates the client and the server from the underlying communication infrastructure and the protocol stack. The protocol stack is replaceable as migration occurs from one implementation of CORBA to another. This provides flexibility for application architectures and simplifies the distributed computing model [Ref: 11]. The client and server roles are dynamic, so an object can act as a client to a published service of another object in one occasion and can itself offer services in another occasion. Figure 6.2 shows the client and server interacting through the ORB.

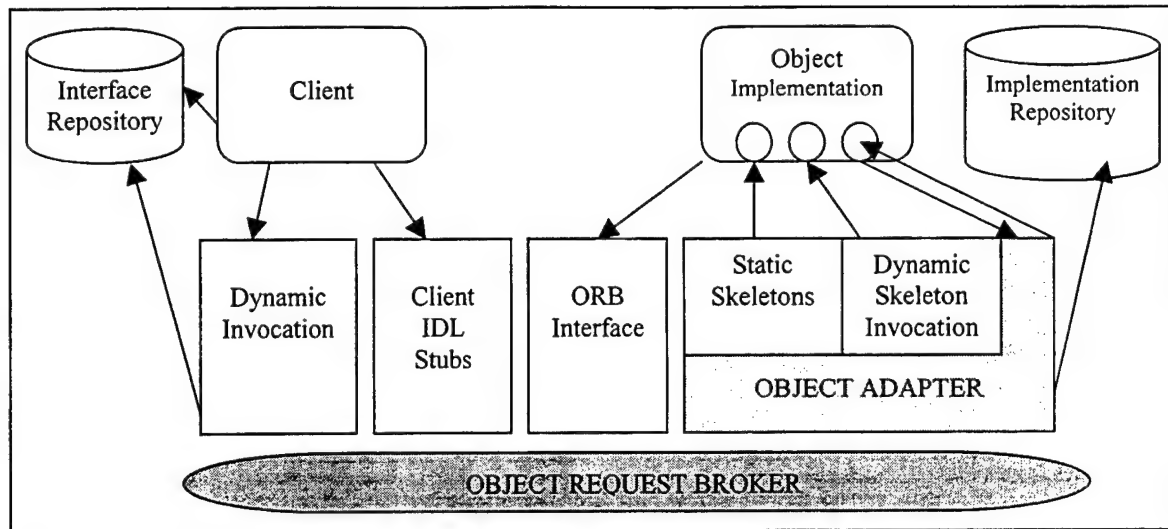


Figure 6.2: The structure of a CORBA 2.0 ORB [Ref: 3]

If we consider the components of the Object Request Broker, the first thing that we need to realize is that CORBA provides both static and dynamic interfaces to the client. Static interfaces are defined at compile-time and provide a robust and efficient way to publish the services provided by an object, in addition they provide fast access at run-time. Dynamic interfaces lack this robustness and speed, but enable the client to discover and use the services of server objects at run-time. The following is a brief discussion of the parts that make up the ORB.

- **Client IDL Stubs:** An interface for an object consists of named operations and the parameters required by those operations. All interfaces in CORBA are defined using Interface Definition Language (IDL). Client IDL stubs are generated by an IDL compiler and provide static interfaces to services provided by an object. A client must have a stub for the services it wants to use on the server. The stub performs the job of packaging parameters into a message format for transfer over the network. This is referred to as *marshalling*.
- **Dynamic Invocation:** This interface allows clients to discover and invoke services offered by an object at run-time.

- **Interface Repository:** IR is a database for storing persistent references to objects that are registered with the ORB. The IR contains enough information for the ORB to locate and activate implementations that correspond to an entry in this database.
- **The ORB Interface:** This consists of services that might be used by an application such as converting an object to a string representation.
- **Object Adapter:** This is a logical set of services that enable the ORB and the implementation of the server object to communicate with each other.
- **Static Skeletons:** Static skeletons are created by compiling the IDL definition of a server object using an IDL compiler for a specific language. Each service supported by a server has a corresponding skeleton that handles the marshalling of the parameters.
- **Dynamic Skeleton Invocation:** These services find the object that offers the service requested by a client by inspecting the parameters and name of the method. This provides maximum flexibility in a rapidly changing environment or an environment with different ORB implementations that have no previous knowledge of each other.
- **The Implementation Repository:** This is a database that can be used to keep track of the server objects and the services they offer.

b. CORBA Services:

CORBA services are provided for the application developer who will provide the implementations of the CORBA objects. CORBA services include services to store, manage and locate objects, to enforce relationships between objects and provide the infrastructure for building licensing and security services.

CORBA SERVICES	
Naming Service	Query Service
Event Service	Licensing Service
Persistence Service	Security Service
Life Cycle Service	Time Service
Concurrency Control Service	Trader Service
Transaction Service	Collections Service
Relationship Service	
Externalization Service	

Figure 6.3: CORBA Services in the object management architecture.

OMG has defined a set of common CORBA services as shown in Figure 6.3. The following is a list of some of the more important services:

- ***Naming Service:*** The naming service is used to associate a human-readable name with a CORBA object reference. The name of the object is bound to the object relative to a naming context. In a naming context each name is unique. Naming service enables CORBA to find another object by resolving the provided name in a naming context.
- ***Event Service:*** This service allows objects to register and unregister their interest in specific events. A common bus named the event channel is used to transfer messages from the generator of an event to objects that have expressed their desire to receive the event.
- ***Persistent Object Service:*** This service provides a set of common interfaces for storing objects in persistent storage. The storage can range in type from a text file to a Relational or Object DBMS.

- ***Life Cycle Service:*** This service includes operations for the creation, copying, moving and deletion of objects from the ORB. Factory objects, which can be used to create CORBA objects, are defined in this service.
- ***Concurrency Control Service:*** This service enables multiple clients to coordinate their access to shared resources. This is achieved by placing a lock on an object to provide atomic access.
- ***Transaction Service:*** Distributed applications need to have certain properties to function properly. The four vital properties are atomicity, consistency, isolation and durability. This service is used for the enforcement of these properties in a distributed system that uses CORBA as its architecture.
- ***Relationships Service:*** This is a general-purpose service for establishing relationships between objects. The expression of a relation in the form of an object makes abstract concepts such as entities and relationships explicitly representable in the distributed architecture. One of the relationship types between objects for example, is containment relationship. This would be represented by a relationship between the container object and the contained objects.
- ***Externalization Service:*** This service defines protocols and conventions for recording the state of an object in a stream of data that can be saved to a file or transported across the network. This process is called *externalization*. The externalized object can be restored by reversing the process, which is called *internalization*.
- ***Security Service:*** This service includes features that can be used to provide a framework for a distributed object system. The issues addressed are the identification and authentication of principals in the system, confidentiality and

integrity of messages sent over the ORB, ensuring availability of resources, providing access control to objects based on the identity and privileges of the requesting object and auditing the actions of a principal.

- **Trading Service:** The functionality of this service is similar to a matchmaking service for objects in the system. An object that provides a service advertises itself by exporting information about the service it provides, the parameters it expects and a reference to itself that can be used by a client to invoke operations on the advertised services.
- **Licensing Service:** Licensing service includes interfaces to protect the intellectual property of developers. Licensing services can be used to control software licenses in a distributed system.
- **Time Service:** This service can be used for the synchronization of different components.

CORBA services are primitive, general and fundamental building blocks that can be used in a distributed object system. They are useful for all kinds of applications and are domain-independent in that they are intended to be reused and specialized by applications [Ref: 11]. An application developer can achieve the functionality desired in an implementation by inheriting from multiple corresponding services. Not all CORBA services are available at this time, but all ORB vendors provide a subset of existing CORBA services.

c. CORBA Facilities:

CORBA facilities are higher-level services that aim to establish application-level interoperability. They provide Interface Definition Language - defined frameworks that are of use to application objects. The interfaces in CORBA facilities are common to multiple domains. The common facilities that are being built by OMG members include facilities to handle user interface management, information

management, systems management and task management. CORBA facilities can reuse services provided by the CORBA services or they can inherit and extend them. Figure 6.4 illustrates the relationship between CORBA services, CORBA facilities and CORBA domains.

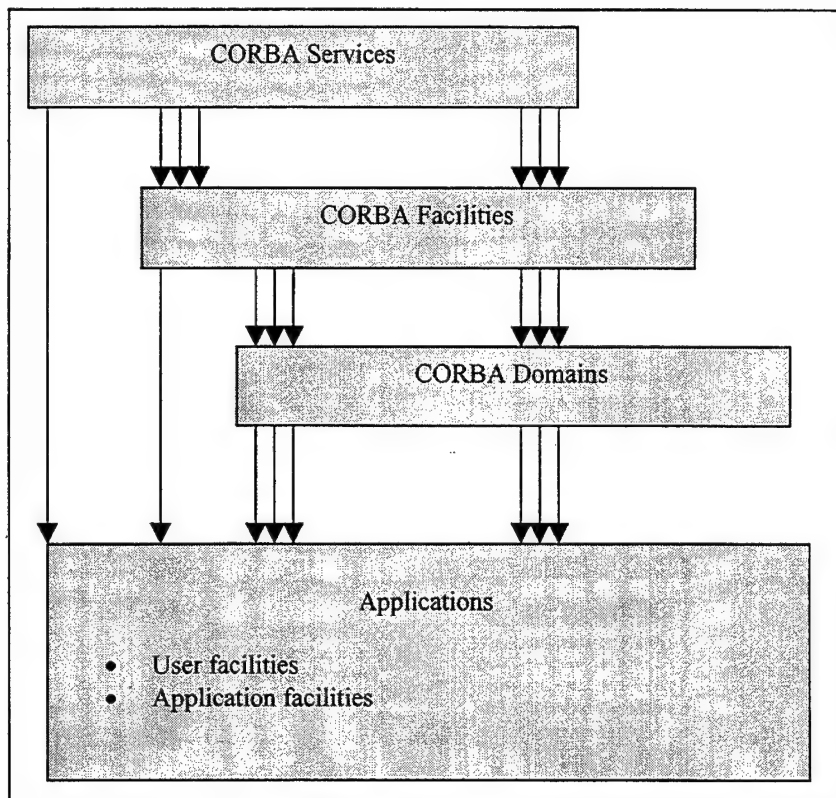


Figure 6.4: Reuse of OMG specifications [Ref: 11]

d. CORBA Domains:

CORBA domains are a business-specific standardization that considers the interoperability needs of specialization areas such as healthcare, manufacturing or telecommunications. CORBA domains do not answer the common needs of multiple domains. This should be handled by CORBA facilities. CORBA domains might use or inherit from the services provided by CORBA facilities or CORBA services as needed.

e. Application Objects:

The last category of objects in the CORBA distributed architecture is the application objects themselves. These objects perform specific tasks for users. They can

use or inherit from the standard interfaces provided by the OMG, including CORBA services, CORBA facilities and CORBA domains or they can provide their own interfaces. Reuse of the services provided by the OMG enables the rapid design and employment of distributed systems and conformity to standards.

B. INTERNET INTER-ORB PROTOCOL

The CORBA 2.0 specification requires ORB vendors to implement the Internet Inter-ORB (IIOP) protocol or to provide half-bridges to it. The goal of this requirement is to ensure communication between different ORB implementations. IIOP makes it possible for a client of one vendor's ORB to invoke operations transparently on an object in a different ORB. The General Inter-ORB protocol (GIOP) defines a set of message formats and common data representations for communication between different ORBs. GIOP is designed to work on any transport protocol. IIOP is a specialization of GIOP that uses TCP/IP as its transport layer.

C. INTERFACE DEFINITION LANGUAGE

Interface Definition Language (IDL) is a standard language used to define interfaces used by CORBA objects. IDL is a part of the CORBA specification and is independent of any programming language. Language independence gives application developers the freedom to choose the language they want to use in the distributed object system. Developers can pick a language that provides better performance, or one in which they have a significant investment. They can even use different languages for different parts of the system. It is also possible to retain and use legacy applications by creating an IDL wrapper for them. IDL mappings exist for a number of languages including C++, Ada95 and Java.

IDL is a declarative language that is syntactically a subset of the ANSI C++ standard. IDL is a very useful tool for software designers, because it separates the implementation of an object from its specification. IDL is used to describe an object's attributes, the services that it provides, the classes it inherits from, the exceptions it raises and the name that can be used to locate the implementation in a distributed architecture.

IDL compilers generate client stubs and server skeletons by processing an IDL file. The structure of an IDL file can be seen in Figure 6.5.

```

module <identifier>
{
    <type declarations>;
    <constant declarations>;
    <exception declarations>;

    interface <identifier> [:<inheritance>]
    {
        <type declarations>;
        <constant declarations>;
        <attribute declarations>;
        <exception declarations>;

        [<op_type>]<identifier>(<parameters>)
        [raises exception[context];
        .
        .

        [<op_type>]<identifier>(<parameters>)
        [raises exception][context];
        .
        .
    }

    interface <identifier> [:<inheritance>]
    .
    .
}

```

Figure 6.5: The structure of an IDL file [Ref: 3]

We will inspect the various parts that make up the IDL file in the following sections.

a. The module:

The module construct in an IDL file is used to group together IDL definitions that are logically related. Modules create a naming scope and are similar in functionality to packages that can be found in various programming languages. You can use the same name for two different interfaces without having a conflict as long as they are in different modules. The following example shows a module with three interfaces.


```

module Bank{
    interface account{ ... };
    interface accountFactory{ ... };
    interface securityManager{ ... };
}

```

The fully qualified name for each interface would be *Module::Interface*, for example *Bank::account*.

b. Primitive Data Types:

CORBA specifies built-in primitive data types that can be used to represent values. Table 6.1 lists the primitive data types supported by IDL.

Type	IDL Identifier	Description
Float point type	float	IEEE single-precision floating point numbers
Float point type	double	IEEE double-precision floating point numbers
Integer type	long	32 bits
Integer type	short	16 bits
Integer type	unsigned long	32 bits
Integer type	unsigned short	16 bits
Char type	char	8 bits
Boolean type	boolean	TRUE or FALSE
Octet type	octet	8-bits
Any type	any	An arbitrary IDL type

Table 6.1: Basic IDL types supported in IDL [Ref: 12]

c. Constructed Types:

Constructed types can be used to combine primitive data types to define user-defined types. The three constructed types supported by IDL are enumerated types, structures and unions.

- **Enumerated Types:** Enumerated types allow the creation of types that can hold one value of a pre-defined list of identifiers. If we were to create a type to represent the days in a weekend, we could create an enumerated type named `weekenddays` in the following way:

```
enum weekenddays (Saturday, Sunday) ;
```

- **Structures:** These can be used to group related primitive and container data types as a single logical construct. Container types will be explained in the next section. The following is a struct that represents a student in a registration database:

```
struct student{  
    string name;  
    string lastName;  
    short studentNumber;  
    boolean graduated;  
};
```

- **Unions:** Unions can be used to represent values of different data types. The value of the type depends on a parameter supplied to the constructor for the union.

```
union balance(short){  
    case 1:  
        short dailyBalance;  
    case 2:  
        long monthlyBalance;  
    case 3:  
        string balanceText;  
};
```

d. Container Types:

Containers are data structures used to store and transmit lists of elements of the same type. Some of these constructs correspond to arrays and other data structures that can be found in programming languages.

- **Arrays:** An array is a fixed-length list of values of the same data type. The following example defines an array that holds ten person structs.

```
person personList[10];
```

- **Sequences:** This is similar to an array, but the size is not fixed. The size of a sequence can be changed dynamically, which provides more flexibility in application design. All values in the sequence must be of the same type. A sequence that is used within an IDL file must be defined by a typedef declaration, such as the following:

```
typedef sequence <person> dynamicPersonList;
```

- **Strings:** Strings in IDL are implemented as a sequence of characters. Their size can be fixed or variable depending on the declaration, such as the following examples:

```
string fixedLengthString[10];  
string variableLengthString;
```

e. Interfaces:

The interface construct is used to describe the services that a CORBA object offers. The services are defined in terms of operations. Operations are similar to methods in object-oriented programming languages or functions in procedural languages. An interface definition includes all information required by a client to use a service provided by the offering object. This information is presented in terms of attributes

belonging to the interface and the parameters that are used by each operation. The interface defines only the signatures. It is not involved in the implementation. Different implementations can be provided for the same interface. The implementation of an object can be changed semantically without breaking the contract. The interface body can contain constant declarations, type declarations, attribute declarations, exception declarations and operation declarations.

An object that is represented by an interface has a state and supports the attributes and operations defined in the interface. Interfaces advertise only the public attributes and operations of an object. An object attribute can be represented as in the following example:

```
attribute string name;
```

Services that are offered by the object are expressed in terms of operations. Parameters for an operation can be defined as in, out or inout depending on the direction of the parameters.

```
void setName(in string name);  
string getName();
```

The following is an example interface declaration in an IDL file.

```
interface Account{  
    const long maximumShare=100000;  
    attribute string accountHolderName;  
    exception TransactionNotAllowed{};  
    boolean deposit(in long amount)  
                    raises (TransactionNotAllowed);  
    double withdraw() raises (TransactionNotAllowed);  
    double getBalance();  
};
```

Interfaces can inherit attributes and methods from other interfaces as shown in the next example:

```

interface Student{
    ...
};
interface Officer{
    ...
};
interface MilitaryStudent: Student, Officer{
    ...
};

```

f. Exceptions:

Exceptions are the means of expressing errors that have occurred while processing an invocation of a method. The exception construct in IDL can contain data types, but no operations. An operation advertises the exceptions that might occur by the “raises” clause. The following IDL file defines an interface with two operations and an exception that might be thrown by these operations.

```

module Bank{
    exception InvalidAccountNameException{string reason};
    interface Account{
        attribute long accountNumber;
        boolean deposit(in double amount)
                        raises (InvalidAccountException);
        double getBalance()
                        raises (InvalidAccountException);
    };
}

```

There are two types of IDL exceptions: user defined exceptions and standard IDL exceptions. Standard exceptions need not be declared, because they can be thrown by any operation. Table 6.2 lists the standard exceptions [Ref: 14].

Exception Name	Description
UNKNOWN	The unknown exception

BAD_PARAM	An invalid parameter was passed
NO_MEMORY	Dynamic memory allocation failure
IMP_LIMIT	Violated implementation limit
COMM_FAILURE	Communication failure
INV_OBJREF	Invalid objet reference
NO_PERMISSION	No permission for attempted operation
INTERNAL	ORB internal error
MARSHAL	Error marshalling parameter or result
INITIALIZE	ORB initialization failure
NO_IMPLEMENT	Operation implementation unavailable
BAD_TYPECODE	Bad typecode
BAD_OPERATION	Invalid operation
NO_RESOURCES	Insufficient resources for request
NO_RESPONSE	Response to request not yet available
PERSIST_STORE	Persistent storage failure
BAD_INV_ORDER	Routine invocations out of order
TRANSIENT	Transient failure
FREE_MEM	Can not free memory
INV_IDENT	Invalid identifier syntax
INV_FLAG	Invalid flag was specified
INTF_REPOS	Error accessing interface repository
BAD_CONTEXT	Error processing context object
OBJ_ADAPTER	Failure detected by object adapter
DATA_CONVERSION	Data conversion error
OBJECT_NOT_EXIST	Nonexistent object

Table 6.2: Standard IDL exceptions

g. Other constructs:

Some other constructs that are worth mentioning are constant declarations, typedef statements, forward declarations and preprocessor directives.

- **Constant Declarations:** Constants can be declared for the following data types: integer, character, boolean, floating point and string. Constants can have interface, module or global scope depending on where they are declared. The following is a declaration of a constant with module scope.

```
module Math{
    const float pi = 3.1415926;
}
```

- **Typedef Declaration:** These can be used to rename an existing data type or create new data types with a user-given name.

```
typedef string[10] AccountName;
```

- **Forward Declarations:** An interface needs to be declared before it can be referenced in an IDL file. Forward declarations can be used to declare the interface without defining it. This enables mutual dependencies and recursive definitions.

```
Interface Professor;
Struct School{
    Professor AcademicAssociate;
};
interface Professor{
    string getName();
};
```

- **Preprocessor directives:** IDL supports the preprocessing features found in C and C++ preprocessors. Developers can access the CORBA services by using the “include” statement in their IDL files, as shown in the following example:

```
//include transactions service
#include <CosTransactions.idl>
```

D. CORBA AND THE WEB

The Internet has become the most significant new medium for communication between and among businesses, educational and governmental organizations, and individuals. As a communications framework, the Internet provides an ideal platform for distributed object applications and therefore fosters their growth. [Ref: 14]

There are numerous technologies in use today, which take advantage of the potential provided by the Internet. The Common Gateway Interface (CGI) protocol has been the dominant model for Client/Server applications using the Internet as a medium. CGI is a slow, stateless protocol that is not suited for distributed object applications. CGI launches a new process to service each client request. Many vendors attempt to overcome the weaknesses of CGI by providing server extensions. These extensions are non-standard and some of them are platform specific. This is not a long-term solution, because the Internet is a heterogeneous environment by nature.

Remote Procedure Call (RPC) is a mechanism that enables programs running on one machine to make calls to functions on another machine that is connected to the Internet. Remote calls are blocking, which means that the calling application can not proceed until it gets the results of the remote invocation. This brings a performance penalty. CORBA method calls can be declared as one way, thus transforming the calls into asynchronous messages. RPC is not object-oriented, so it cannot take advantage of features like encapsulation, inheritance and polymorphism.

There are other competing distributed object models that are fully object-oriented. The two most notable ones are Java's Remote Method Invocation (RMI) by Sun Microsystems and Distributed Component Object Model (DCOM) by Microsoft. RMI does not provide language-neutral messaging services. An object written for RMI needs to be written in Java and can operate only with objects that are implemented in the same language. RMI does not support dynamic invocations and interface repositories. It does not define the protocols for services like transactions and security. DCOM has serious limitations as well. CORBA objects have unique and persistent references and

they have state, where DCOM objects do not maintain their state between connections. This creates a problem in an environment like the Internet where there are numerous faulty connections. In addition, it is very difficult to configure and run DCOM applications on non-Windows platforms. DCOM does not support a universal naming service, which severely limits scalability.

The limitations of alternative approaches pinpoint CORBA as the leading model for providing the component-based Internet-enabled application architecture. We used CORBA to implement our Enterprise Information System prototype for the Naval Security Group. The ORB implementation that we used was Visigenic Visibroker 3.3. The design process and implementation issues will be discussed in Chapter VII.

VII. IMPLEMENTATION OF A WEB BASED CLIENT/SERVER SYSTEM USING CORBA AND JDBC

A. ORB SELECTION

A variety of ORB implementations are available, each with different features. The ORB implementation that we used for our EIS prototype was Visigenic Visibroker 3.3. The main reason for the selection was the stability of this product and the fact that the Java binding was completely written in Java, which meant that a client with access to the server via the web could download the ORB on the fly. Although the ORB implementation was by Visigenic, the application was constructed using ORB-independent services. This feature gave us the ability to run the application on any CORBA-compliant ORB implementation with only minor modifications to the code.

B. APPLICATION DESIGN PROCESS WITH VISIBROKER

To illustrate the application design process, we will first describe a simple distributed system to fully understand this process. The following are the steps that we need to follow to design our sample system.

- 1) ***Make an analysis of the functionality required of the system and divide the functionality into objects:***

In our sample Client/Server system we will build a Client/Server system that provides quotes to the clients on demand. The server object will listen to the client and respond to client requests with a randomly chosen quote from its database. After deciding the services that the server should support, we need to write a specification for the server object using Interface Definition Language (IDL). The IDL file can be created using any text editor. This file lists the public attributes and the public services of the objects defined for the system. The information contained in the IDL file acts as a contract between objects and their potential clients. The client need not be concerned how the

object is implemented. The following is the IDL file for the QuoteServer object.

```
module Example{  
    interface QuoteServer{  
        string getQuote(in string name,  
                        in string day, in string month);  
    };  
}
```

The file is saved as Example.idl.

2) ***Compile the IDL file with an IDL compiler to create the client stub code and server skeleton code in the desired implementation language:***

For our prototype we used the Visibroker idl2java compiler. We entered the following in the command line in the directory containing Example.idl.

```
prompt> idl2java Example.idl -no_comments -no_tie
```

The `no_comments` flag indicates to the compiler that we do not wish the creation of comments. The `-no_tie` flag indicates which type of server implementation we prefer. Different server implementation strategies are explained further in this chapter.

3) ***The compiler creates a number of files that are used by the ORB to process remote method invocations:***

One of the files generated by the compiler is an example file that contains the constructors and the method implementations that have empty bodies. The developer can use this file as a starting point when providing implementations for the objects. In our example, the compiler generated the following files:

- ***QuoteServer.java***: This is the public Java interface that the server object must implement. The module name is used as the package name and all operations become public method declarations. All interfaces extend the root object in the ORB implementation.

```
package Example;

public interface QuoteServer
    extends org.omg.CORBA.Object {

    public java.lang.String getQuote(
        java.lang.String name,
        java.lang.String day,
        java.lang.String month);
}
```

- ***QuoteServerHelper.java***: This is a Java class that provides helper functions for clients that want to use the services of a QuoteServer object.
- ***QuoteServerHolder.java***: Java allows parameters to be passed only in the *in* mode. This Java class enables the passing of *out* or *inout* parameters.
- ***_st_QuoteServer.java***: This is the stub class that the client uses to invoke the operations defined in the IDL file. It implements the interface QuoteServer. This class has methods to perform the marshalling and unmarshalling of parameters.
- ***_QuoteServerImplBase.java***: This class is the server skeleton class that implements the QuoteServer interface. It is responsible for unmarshalling the arguments for the QuoteServer object. The implementation provided by the developer for the server must extend this class.

- *_example_QuoteServer.java*: This is the example class that can be expanded to provide the implementation for the QuoteServer object. It contains the constructors and the empty method definitions for the operations defined in the IDL file. The example class generated by the idl2java compiler is as follows:

```
package Example;
public class _example_QuoteServer extends
    Example._QuoteServerImplBase {
    public _example_QuoteServer(java.lang.String name) {
        super(name);
    }
    public _example_QuoteServer() {
        super();
    }
    public java.lang.String getQuote(
        java.lang.String name,
        java.lang.String day,
        java.lang.String month) {
        // IMPLEMENT: Operation
        return null;
    }
}
```

4) *Write the implementation for the server :*

The server must provide the implementations of the methods advertised in the interface. This is accomplished by defining a Java class that has public method definitions corresponding to the operations published in the interface and variables with accessor methods that correspond to the attributes in the interface. The developer can have extra methods that help the business logic, but these will not be accessible by the client. There are two approaches to implementing the server: implementation by delegation and implementation by inheritance. The client code is not effected by the choice of the server

implementation model. The implementation class extends the skeleton class in the inheritance approach and implements the operations interface that is generated by the idl2java compiler in the delegation approach. We will use the inheritance approach in our example and write the following class definition:

```
package Example;

public class QuoteServerImpl extends
    Example._QuoteServerImplBase {
    private DBHelper QuoteHelper;
    public QuoteServerImpl(java.lang.String name) {
        super(name);
        QuoteHelper = new DBHelper();
        QuoteHelper.connect();
    }
    public QuoteServerImpl() {
        super();
        QuoteHelper = new DBHelper();
        QuoteHelper.connect();
    }
    public java.lang.String getQuote(
        java.lang.String name,
        java.lang.String day,
        java.lang.String month){

        try {
            return QuoteHelper.getQuote(name, day,
                                         month);
        } catch (Exception e){
            System.out.println("System Exception in
                               getQuote");
            return null;
        }
    }
}
```

}

The QuoteHelper object has methods to connect to a database server and run a query to receive the “quote of the day”. The implementation of this object is not provided as it’s not directly related to this chapter.

The server object requires a class with a main function to act as a driver. The main function will initialize the ORB, create a basic object adapter, create the QuoteServerImpl object, notify the Basic Object Adapter (BOA) that the server object is available and wait for incoming requests. The following is the driver class for our example:

```
class QuoteServer {
    static public void main(String[] args){
        try{
            // Initialize the ORB
            org.omg.CORBA.ORB orb =
                org.omg.CORBA.ORB.init(args, null);

            // Create the basic object adapter
            org.omg.CORBA.BOA boa = orb.BOA_init();

            // Create the QuoteServer object
            QuoteServerImpl quote = new
                QuoteServerImpl();

            // Notify the BOA that the object is available
            Boa.obj_is_ready(quote);

            // Wait for requests
            Boa.impl_is_ready();

        } catch(org.omg.CORBA.SystemException e){
            System.err.println(e);
        }
    }
}
```

}

5) *Write the client implementation that utilizes the services provided by the server object:*

The client need not be written in the same language as the server. The client application needs to obtain an object reference in order to make calls. There are various ways the client can obtain this reference:

- As a parameter or result of other IDL operations
- Using the Visigenic proprietary bind() method and smart agent
- By reading an IDL defined object representation (IOR) converted to a string and stored in a public location on the server
- Resolving a name within a CORBA Naming Service

For our example we will implement the client using the bind() method. The client in our EIS implementation will use the CORBA Naming Service to obtain a reference to the server, due to the ORB-independent nature of this service. The following is the code for the client for our QuoteServer object:

```
class QuoteClient{
    String quoteOfTheDay;
    public static void main(String args[]){
        try{
            // Initialize the ORB
            org.omg.CORBA.ORB orb =
                org.omg.CORBA.ORB.init(args,null);

            // Bind to the QuoteServer object
            Example.QuoteServer quoteServer =
                Example.QuoteServerHelper.bind
                    (orb, quoteServer);

            // Get the quote of the day
```



```

        quoteOfTheDay = quoteServer.getQuote
            (args[0], args[1], args[2]);

        // Print the quote to the screen
        System.out.println(quoteOfTheDay);
    } catch(org.omg.CORBA.SystemException e){
        System.err.println (e);
    }
}
}

```

6) *Compile the client and the server code with the language compiler:*

Since we are using Java for the implementation language, we will use the javac compiler that comes with the JDK1.1.6 distribution. In order for the compilation to be successful, the client and server files must be in the same directory as the QuoteServer.idl file. The reason for this is that the client and server classes require access to the classes in the Example package, which is created as a subdirectory of the current directory by the idl2java compiler. The server class is compiled with the following command:

```
prompt> javac QuoteServer.java
```

The client class is compiled with the following command:

```
prompt> javac QuoteClient.java
```

7) *Start the server:*

Inprise Visibroker 3.3 requires at least one OSAgent service to be running. The OSAgent provides location services when the objects are using the proprietary binding mechanism to locate each other. It can be started manually or registered as a Windows NT service, if the system is running on

Windows NT. The OSAgent is started manually with the following command:

```
prompt> start osagent -c
```

Next the server program needs to be started with the following command:

```
prompt> java -DOAid=Tsession QuoteServer
```

The "Tsession" flag is required to disable the thread pooling capability of Visibroker. This feature is used to improve performance when multiple clients share the server.

8) *Start the client application:*

The client application needs to be located in the same directory as the stub files. Enter the following command:

```
prompt> java QuoteClient client_1 1 1
```

The design steps for this example are illustrated in Figure 7.1.

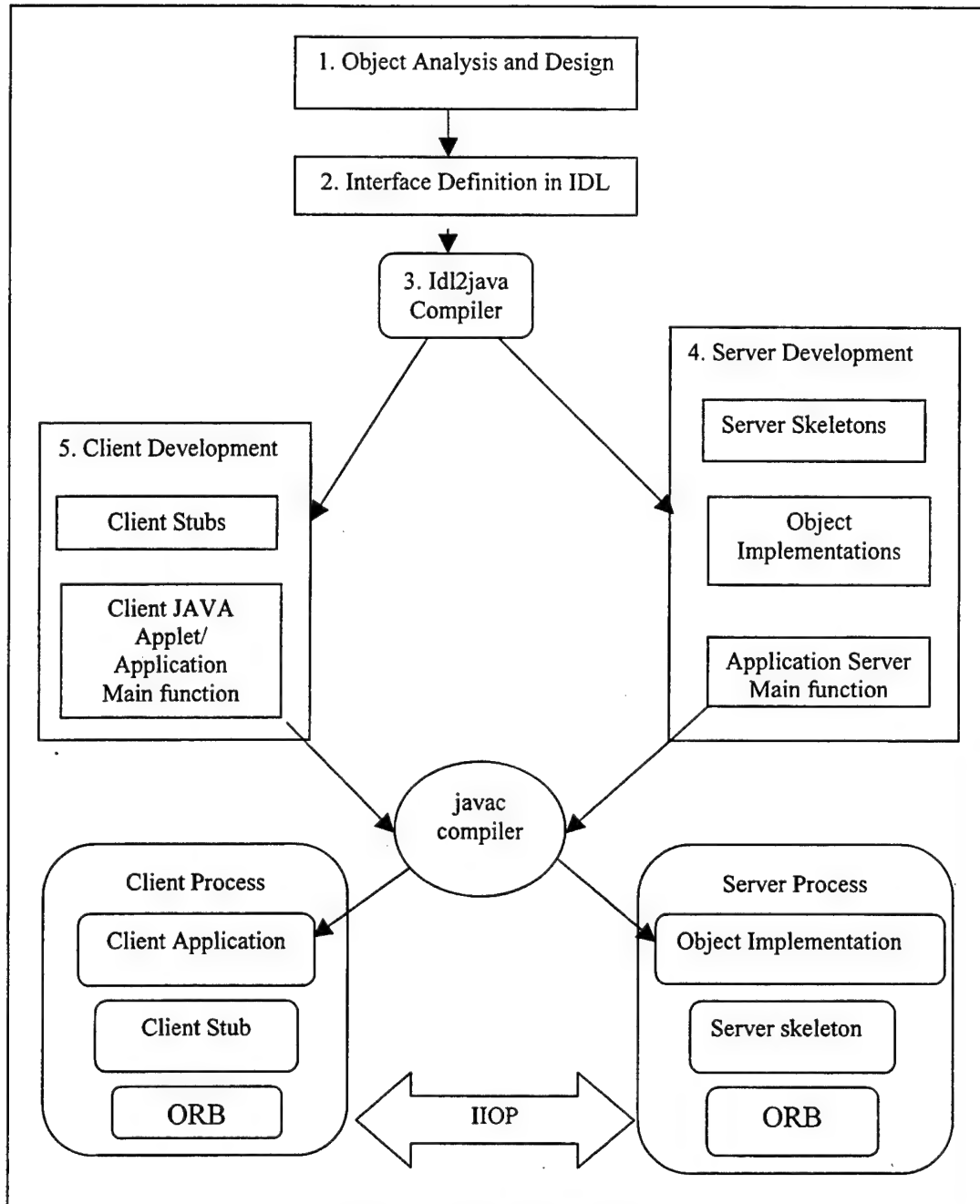


Figure 7.1: IDL design steps

C. APPLETs AND CORBA

The web browser has emerged as a universal front-end for distributed systems. Internet/Intranet technology provides organizations with a cost-effective and reliable medium for building distributed systems. Although HTTP is the most widely used protocol in use, any protocol that is built on TCP/IP protocols can be used for web-based

applications. CORBA is a protocol which solves the performance, scalability and managability problems that application developers face when building distributed systems. The mediator between CORBA and a browser is the Java applet. Applets are mobile applications that are designed to run within a browser. A browser controls the life cycle of an applet. The applet is sent to the browser by embedding it inside a Hypertext Markup Language (HTML) page. An APPLET HTML tag is used to define the applet. The APPLET tag must include information about the name of the Java class and the size of the applet on the HTML page. The CODE parameter defines the name of the Java class that should be downloaded, while the WIDTH and HEIGHT parameters define the size of the applet within the HTML page. Optional parameters are the CODEBASE parameter, which indicates the directory that hosts the Java class file, if it is not in the same directory as the HTML page, and the PARAM parameter, which is used to define the parameters required by the applet. Below is an HTML page that can be used to download a Java applet, which uses the CORBA Internet Inter ORB protocol (IIOP) to interact with the server:

```
<HTML>
<HEAD>
<TITLE> CORBA APPLET </TITLE>
<BODY>
<applet CODE="Gui.class"    WIDTH="795" HEIGHT="594"
                                HSPACE="100"  ALIGN="TOP">
<PARAM NAME="org.omg.CORBA.ORBClass"
                                VALUE="com.visigenic.vbroker.orb.ORB">
<PARAM NAME=ORBservices VALUE=CosNaming>
<PARAM NAME=SVConameroor VALUE=JavaCorba>
</applet>
</BODY>
</HTML>
```

The first parameter in this case is used to force the browser to load the ORB classes from the web server. The other two parameters are used by the applet to indicate

that the applet requires the CORBA Naming Service to find the server object and that the root-naming context is JavaCorba. The CORBA Naming service will be explained in the next section. Figure 7.2 shows how HTTP and CORBA protocols are used to provide interaction between the server and the client.

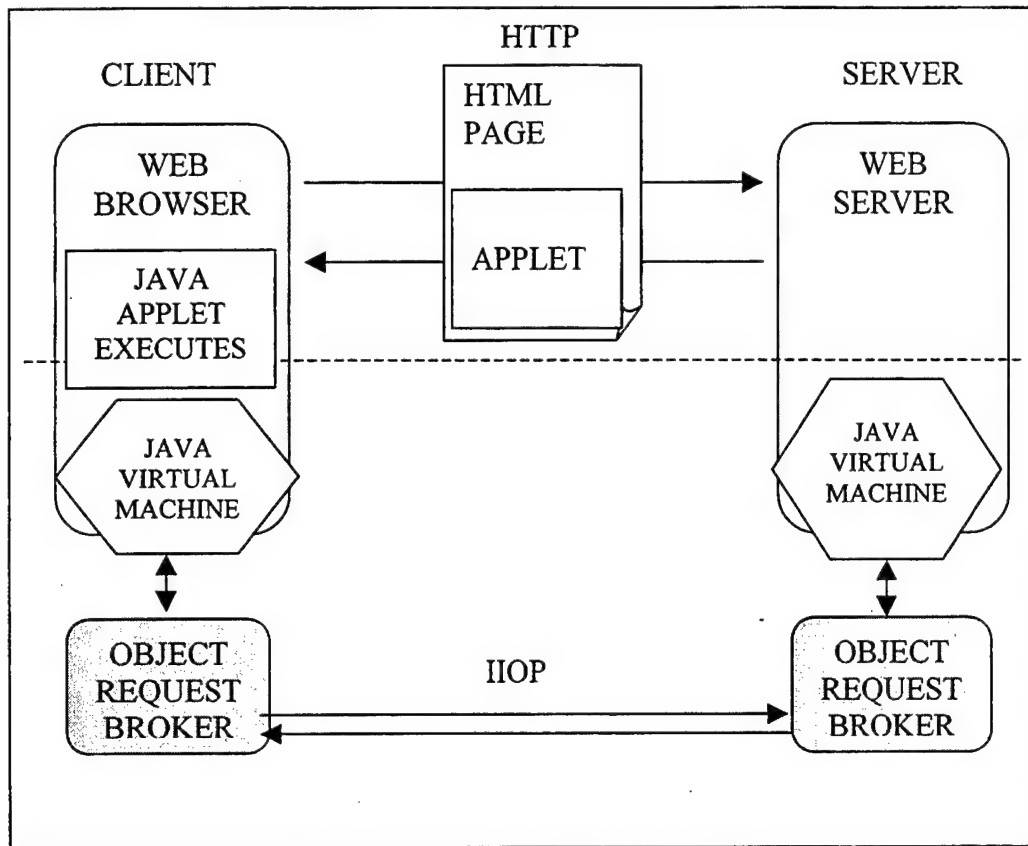


Figure 7.2: Applet-CORBA interaction

The system is initialized by placing the applet and other Java class files on the web server. The web browser will request the page that embeds the applet when the user enters the Uniform Resource Locator (URL) name for the page. The web server will return the web page to the browser using the HTTP protocol. The applet and the Object Request Broker classes will then be sent with the page. The browser will recognize the APPLET tag in the page and activate the Java virtual machine. The Java virtual machine will create an instance of the applet and in the process bind the applet to the server by using IIOP. When the user requests a service from the server, the applet will create the proper CORBA request and send the request to the server using IIOP. The server will return the results of the request using the same protocol. The applet will then translate

the results and display them to the user. The only problem in this scenario is the security model that has been built in Java. The security model in Java allows applets to connect only to the host system from which they have been downloaded. This is called “The Java Sandbox”. The main reason for this limitation is that once an applet is run on a machine inside an organization’s firewall, it has the privileges of the user on whose behalf it is executing. The applet can access private information and transmit it via a TCP/IP connection without the user’s knowledge. This restriction seriously limits the capabilities of a distributed system that uses applets. Fortunately with the new Security model in JDK1.1 it is possible to grant applets fine-grained access rights beyond the “Sandbox Model”. This is achieved by digitally signing the applets. In Visibroker 3.3 a proprietary mechanism has been developed to counter the limitations placed on the applet. A proxy server named “Gatekeeper” enables the forwarding of messages from the client applet to a server and then returning the results from the server to the client. The Gatekeeper acts only as a forwarder of IIOP packets, therefore it is vendor-neutral. The client may be using Visibroker, while the server uses another ORB implementation. Figure 7.3 illustrates the operation of the Gatekeeper.

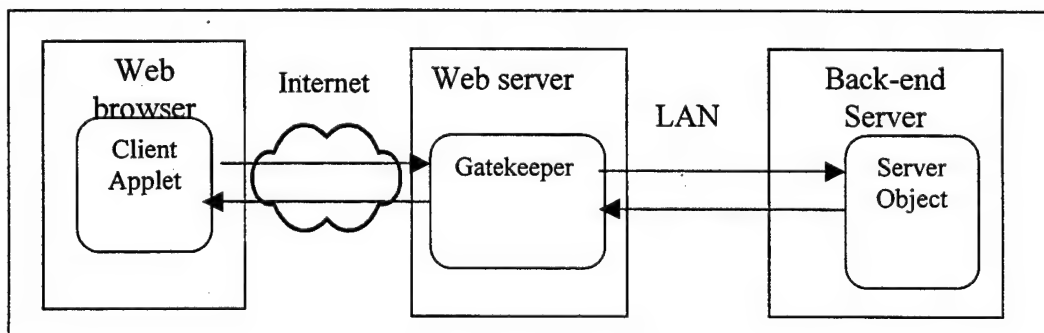


Figure 7.3: Operation of the Gatekeeper

D. CORBA NAMING SERVICE

Every CORBA compliant ORB must support the NAMING Service defined in the CORBA Services Specification. Visibroker currently implements the following Object Services:

- Naming Service

- Event Service
- Security Service over Secure Socket Layers
- Transaction Service

We did not use the bind utility provided by Visibroker 3.3 in our prototype. Instead we used the vendor neutral CORBA Naming Service. The problem associated with using the bind utility in Visibroker is non-portability. All components in the distributed system must have Visibroker as their ORB and the ORB can not be changed without taking out all binding method calls and replacing them with Naming Service method calls. The Naming Service has a very flexible structure that allows developers to create their own naming hierarchies.

Every CORBA object can be located by using a series of bindings that end in a simple name. The Naming Service forms a hierarchy of name components in the form of a tree structure. Each name component holds a reference to an object. This object can be another name component or the instance of the object that is sought. Each name component forms a name context. The Naming Service traverses the name components until it finds the instance of the object to which it returns a reference or ends its traversal without finding the object and throws an exception. This process is called *resolving* a name. To create an association between a name and an object is called *binding*. The Naming Service maintains a database of bindings between names and object references. The binding and resolving process is illustrated in Figure 7.4.

- ***Binding an object to a name:*** In order to bind a name to an object the first task must be to acquire a reference to the Naming Service. This is accomplished as follows:

```
org.omg.CORBA.Object nameServiceObj =
    orb.resolve_initial_references(NameService);

org.omg.CosNaming.NamingContext nameService =
    org.omg.CosNaming.NamingContextHelper.
        narrow(nameServiceObj);
```

The next step is to construct the name to which the object will be bound:

```
NameComponent nameRoot =  
    new NameComponent(QuoteServer, "");  
  
NameComponent[] QuoteServerName = {nameRoot};
```

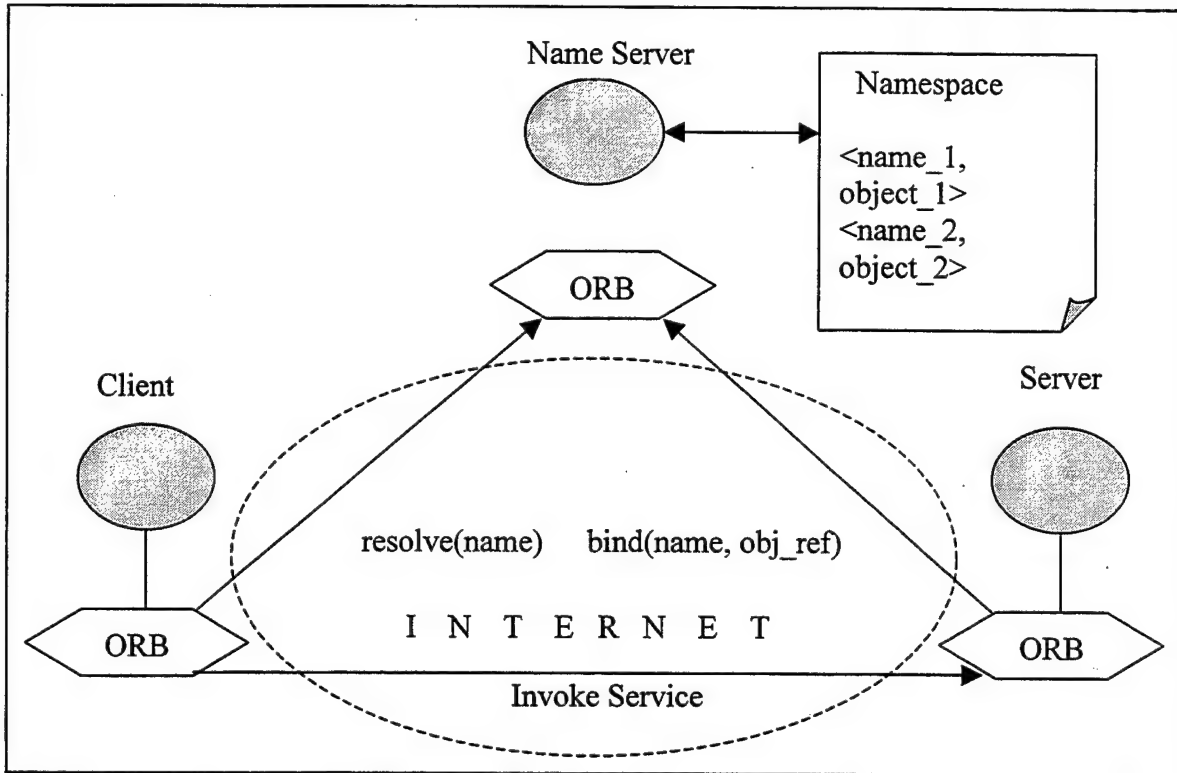


Figure 7.4: How Clients and Servers interact with a Name Server [Ref: 3]

Next the bind method of the Naming Service should be called. We preferred to use the rebind method in order to avoid exceptions if the name was previously bound.

```
NameService.rebind(QuoteServerName, QuoteServer);
```

- **Resolving a name to obtain an object reference:** The client of the Naming Service needs to obtain a root naming context, then traverse through the naming context to obtain a reference to the object for which it is searching.

The client obtains a reference to the root naming context with the following code:

```
org.omg.CORBA.Object rootRef =  
    orb.resolve_initial_reference(NameService);  
  
NamingContext rootContext =  
    NamingContextHelper.narrow(rootRef);
```

The next step is to create the name of the object:

```
NameComponent[] objName =  
    {new NameComponent(QuoteServerName, " " )};
```

Finally, the client asks the Naming Service to resolve this name and return a reference to the object bound to that name:

```
Example.QuoteServer quoteServer =  
    Example.QuoteServerHelper.narrow  
        (nameService.resolve(objName));
```

- ***Running the Naming Service with Visibroker:*** The Visibroker implementation of the Naming Service can be executed with the following command. This service must be running on the server to allow clients to access objects using the Naming Service.

```
prompt> vbj -DORBServices=CosNaming  
            -DSCVnameroot=JavaCorba  
            -DJDJrenameBug  
            com.visigenic.vbroker.services.CosNaming.  
                                                    ExtFactory  
            JavaCorba namingLog
```

By using the Naming service, the client can be completely vendor-independent. This means that the ORB on the server can be replaced without changing a single line of client code.

E. DESIGN AND IMPLEMENTATION OF THE EIS PROTOTYPE

1. Design

A logical plan should be followed in the design process of a distributed system. The first phase should be to identify the objects that will be used in the system. Different methodologies can be employed to work from the problem statement and come up with a division of functionality between objects that make up the system. The content, behavior and interaction of objects are described in the information architecture of the application. A suggested methodology for developing an information architecture in a distributed system is as follows, the reader should consult to the reference for a more detailed explanation of the methodology [Ref: 12]:

- Perform domain analysis
- Model the domain with UML use cases, which present the functionality of the system when combined together
- Develop an object model from use cases
- Develop behavioral models, state diagrams, or object interaction diagrams for behavior intensive or time-critical areas of the system
- Partition the object model into subsystems and allocate the subsystems to CORBA servers
- Design the interface with IDL

The system that we built needed to satisfy the requirements for the NAVSECGRU to establish and maintain a central database that could be accessed from anywhere in the world. The end user would have computer competency ranging from expert to novice. An assumption that the end user would be proficient in using Windows based applications was made. The specification for the database was not concrete, so that the database and its relational tables could be designed or modified according to the organization's needs. The system would even be able to swap the back-end DBMS

without requiring extensive modifications. We took a subset of the information stored in a typical Navy command and came up with the following use cases:

- Enter personnel information
- Modify personnel information
- View personnel information
- Enter budget and supply information
- View budget and supply information
- Enter command information
- Modify command information
- View command information
- Enter operational information
- View operational information

The next step in the design phase of our EIS was to produce the object model, which would provide a basis for the objects in the system and the interaction between them. The classes with specific responsibilities have been identified in this phase. Figure 7.5 shows the simplified EIS class diagram. The Unified Modeling Language (UML) notation, which has been accepted as the standard object modeling technique by the OMG, was used. The reader can find further information regarding UML in Appendix A.

We then defined the interaction between the objects that made up the system, with the use of sequence diagrams. Sequence diagrams give a clear picture of the interaction between objects in a specified scenario. Each use case may be broken down into a number of scenarios and each scenario can then be represented with a sequence diagram. The following scenarios were defined as an example for the “View operational information” use case:

- View all sea missions in a given period of time
- View sea missions for a type of platform in a given period of time
- View sea missions according to their types in a given period of time
- View sea missions in an exercise

- View sea missions executed by a specific platform in a given period of time
- View all air missions in a given period of time
- View air missions flown by a specific platform
- View air missions according to their types in a given period of time
- View air missions flown in an operational area in a given period of time
- View air missions flown by a squadron

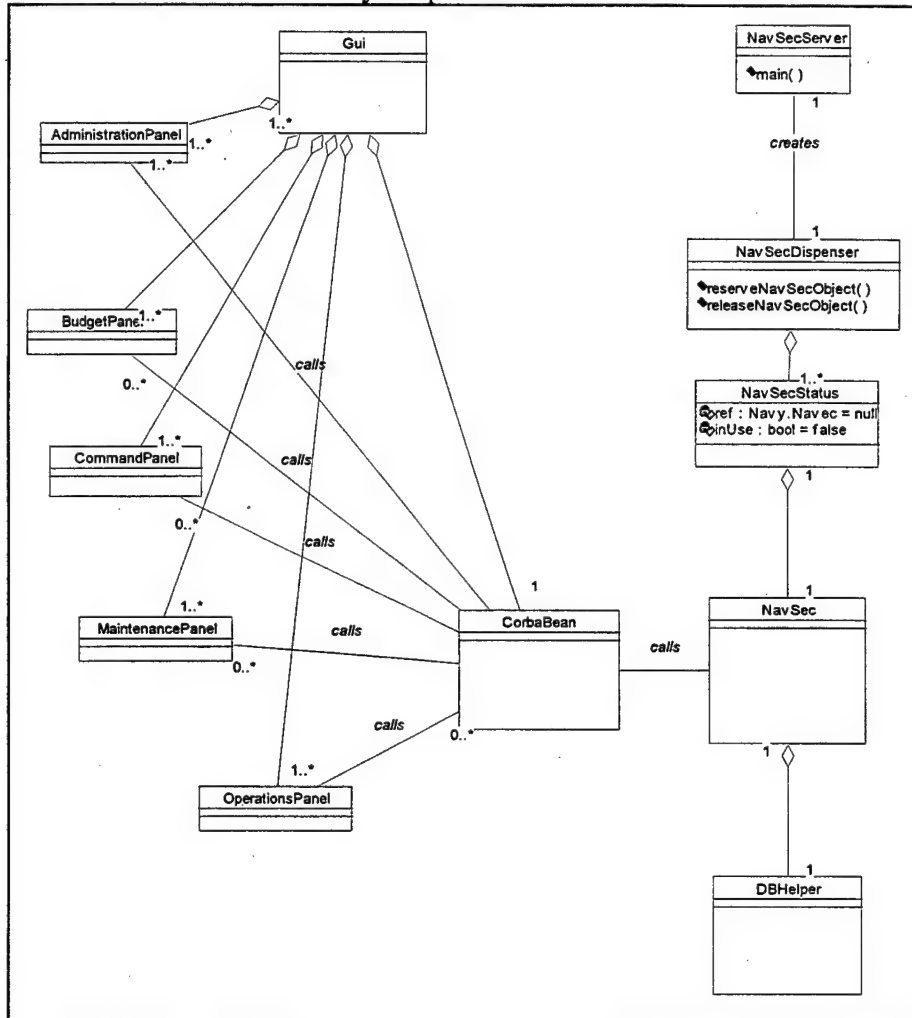


Figure 7.5: EIS class diagram

The user of the EIS is the performer for each of these scenarios. The user directly interacts with the EIS in order to achieve a specific goal. Each goal is accomplished through the successful completion of a series of actions. The sequence of interactions that can occur under certain conditions makes up the scenario. A sequence diagram shows the steps of interactions and the resulting state. Sequence diagrams can be used to

further specify the behaviors that should be expected of an object. We used sequence diagrams to define the services that each object in the system should provide. Figure 7.6 shows the sequence diagram for the “Add Air Mission” scenario as an example.

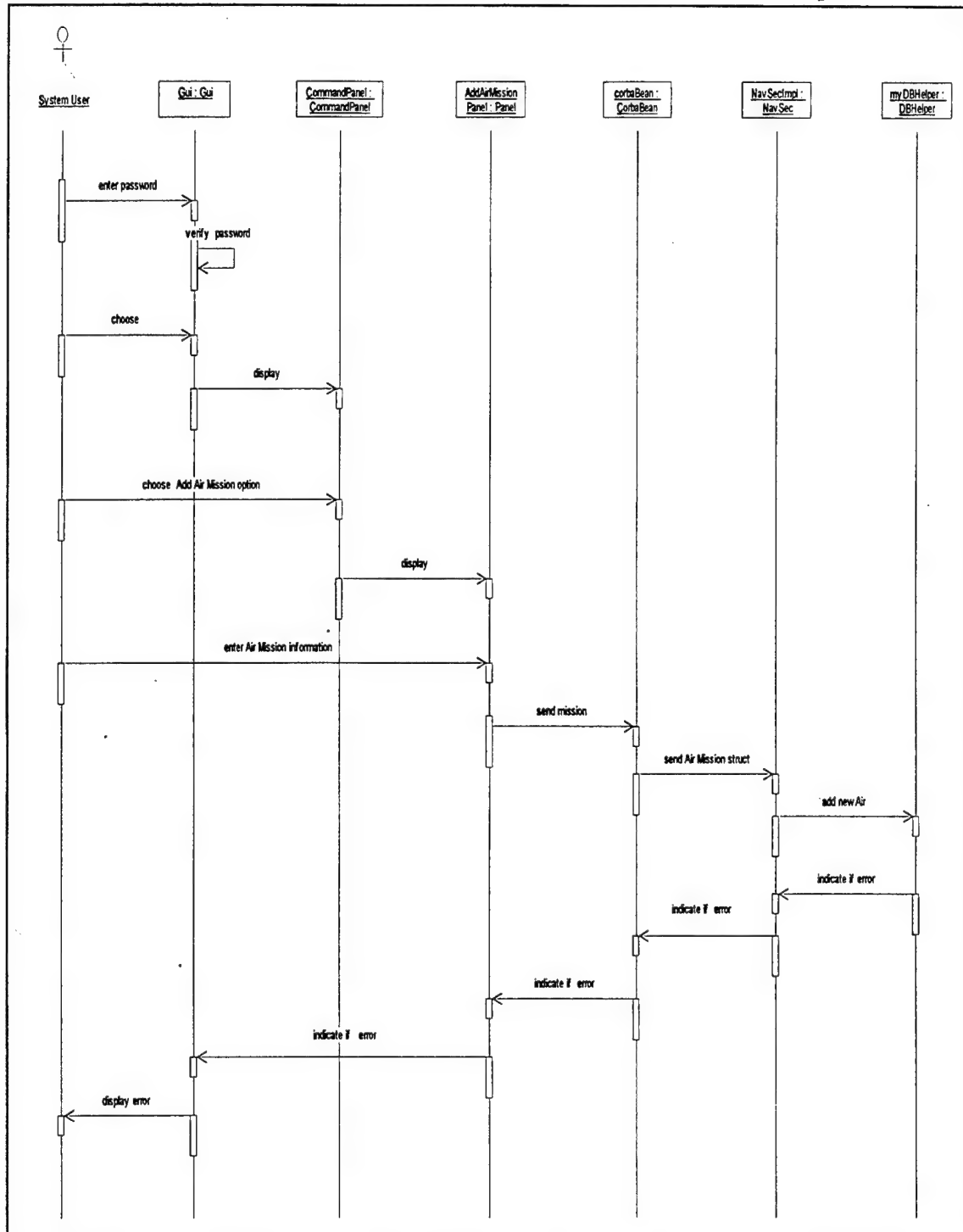


Figure 7.6: “Add Air Mission” sequence diagram

At the end of the design phase we had a very concrete picture of the objects and the functions that should be supported by each object. The next step was to write the Interface Definition Language (IDL) specification that would serve as a contract between the client and server parts of the system. We provided definitions for the data structures and the interfaces for the CORBA server objects within the system in the IDL file NavSec.idl. We will cover different parts of this file as we continue with the definition of objects in the system. The complete file listing can be found in Appendix B.

2. Implementation

Our EIS prototype is built on a three-tiered Client/Server system. The benefits of this choice have been covered in the Client/Server chapter. The client interface is implemented using the JApplet class of the Java Swing package. This is a “thin” client implementation, which means that the business logic is implemented on the middle tier vice the front-end. JApplet class extends the Applet class in the Abstract Window Toolkit (AWT) package. We chose to use Swing, because it provided the developer with a wide range of user interface widgets that reduced development time. Since none of the popular browsers supported Swing at the time of development, the installation of the JDK1.1 plug-in is required for each client. If a user that has not installed the JDK1.1 plug-in visits the page hosting the client interface, the user is automatically directed to the Javasoft site where the plug-in can be downloaded. When the plug-in is installed, the client then has access to all of the Swing package classes. The client JApplet class requires loading of the ORB by defining it as a parameter. The Naming Service is also defined as a parameter and requires loading. The client JApplet uses the Naming Service, so that the distributed system is as ORB-independent as possible.

During operation of the EIS, the client invokes methods on the CORBA server objects in the middle tier. These server objects interact with objects that use JDBC to execute queries on a relational DBMS. This interaction constitutes the third tier. Only server objects can interact with the back-end database, which enforces the business rules of the system within the middle tier and keeps all information stored in the database in a consistent state. Figure 7.7 shows how the EIS prototype is divided into three tiers.

We will discuss in detail each of the three tiers that make up our EIS Client/Server system and discuss the objects that operate within each tier in the following sections.

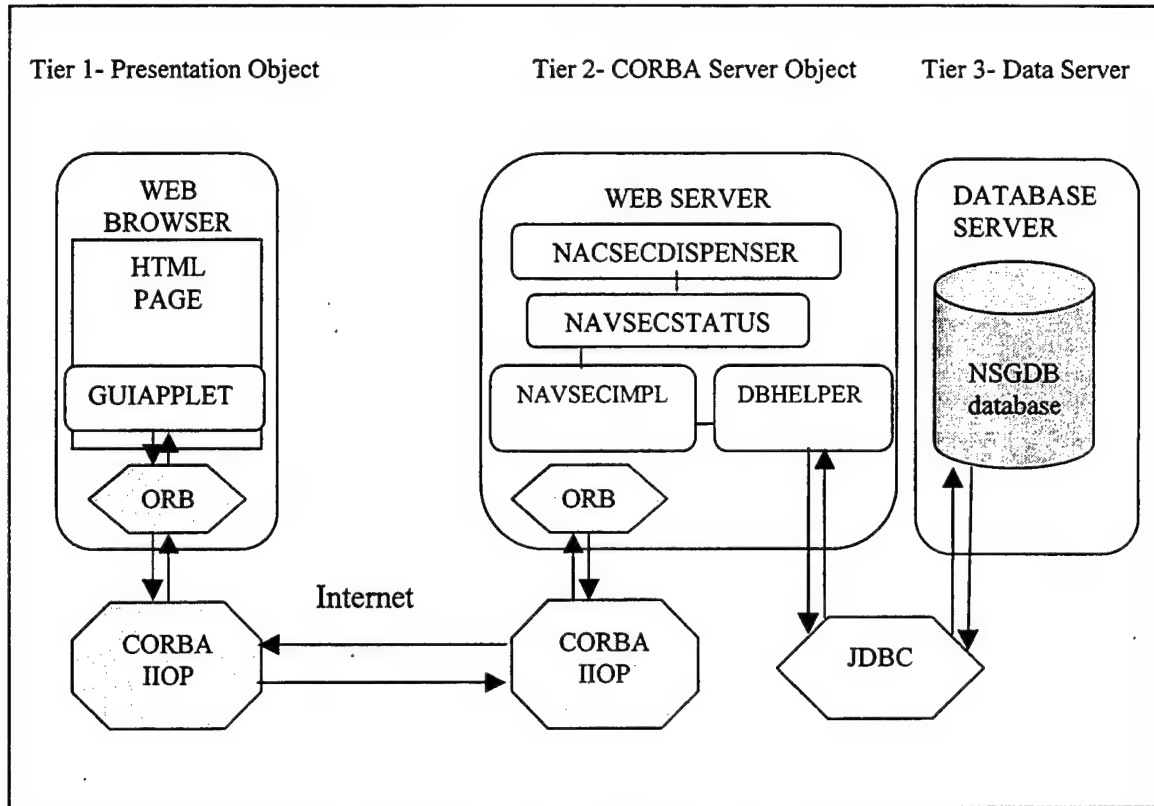


Figure 7.7: Objects in the EIS architecture

3. Data Server

The data server for the EIS prototype consisted of a database instance on a SQLServer 6.5 database server. The database instance was named NSGDB after “Naval Security Group Database”. This instance consisted of six logical subdivisions and twenty-three tables. The structure of the NSGDB database is outlined in Figure 7.8.

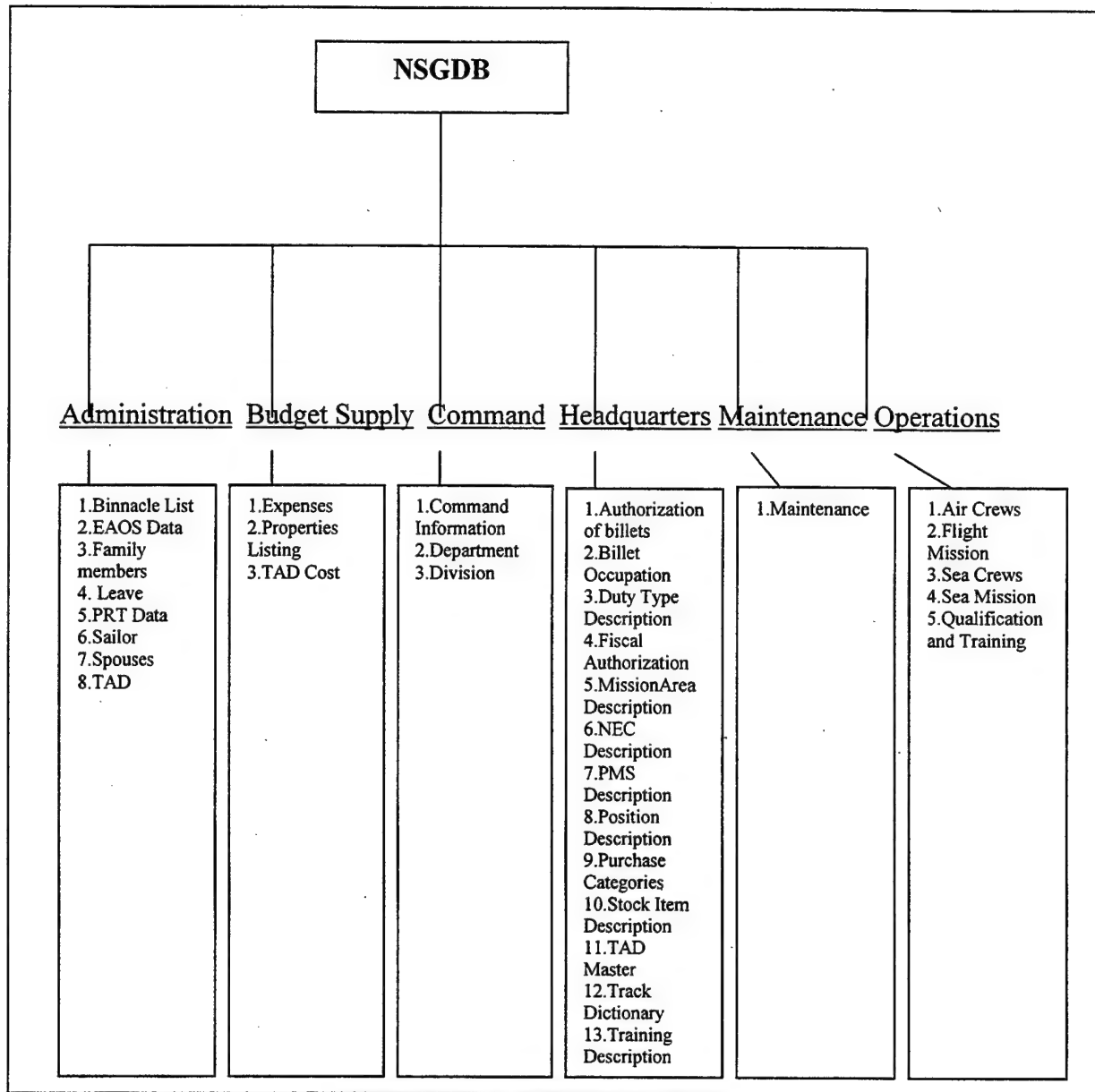


Figure 7.8: NSGDB Database structure

A helper class called DBHelper.java was created to access, query and manipulate this database. The Connect Software FastForward Microsoft JDBC driver was used for access to the NSGDB database. The following is a brief description of the DBHelper class:

- ***DBHelper.java***: The IDL file does not advertise this class. It is a local class that provides methods for connecting to the back-end database, executing SQL queries with user defined parameters and disconnecting from the database. DBHelper objects establish a connection to the NSGDB database instance when they are created and maintain this connection in order to avoid the penalty of establishing a new connection with the DBMS whenever a new client makes a request. The following code shows the method that is used to establish a connection with the database:

```

public void connect (String address, String db,
                    String id, String pw) throws Exception
{
    try{
        // Load the FastForward JDBC driver
        Class.forName(connect.microsoft.
                      MicrosoftDriver);

        // Create the address for the database
        String url = "jdbc: ff-microsoft://" +
                    address + ":1433/" + db;

        //Get a connection to the database
        Con = DriverManager.getConnection(url ,id,
                                         pw);

    } catch( Exception e){
        System.err.println(System exception in
                           connect);
        System.err.println(e);
    }
}

```

Once the connection with the database is created, it is possible to run queries on the database and return the results to the requesting object. The

method definition provided as an example on the next page is used to insert information about a new department into the database:

```
public boolean addDepartment( Navy.departmentStruct
                             departmentData)
                             throws Exception
{
    try {

        pstmt = con.prepareStatement(
            " INSERT INTO DEPARTMENT( "+
            " COMMAND_ID, DEPARTMENT_ID, TITLE ,"+
            " DEPARTMENT_HEAD_SSN, "+
            " DEPARTMENT_CHIEF_SSN, "+
            " TELEPHONE, EMAIL )"+
            " VALUES ( ? ,? ,? ,? ,? ,? ,? ) ");

        pstmt.setString( 1, departmentData.commandId);
        pstmt.setString
            ( 2, departmentData.departmentId);
        pstmt.setString( 3, departmentData.title);
        pstmt.setString( 4, departmentData.headSsn);
        pstmt.setString( 5, departmentData.chiefSsn);
        pstmt.setString( 6, departmentData.telephone);
        pstmt.setString( 7, departmentData.email);
        pstmt.executeUpdate();
        pstmt.close();
        con.commit();
        System.out.println("Query run successfully");
        return true;

    } catch(Exception e) {

        System.err.println
            ("System Exception in addDepartment");
        System.err.println(e);
        throw e;
    }
}
```

The DBHelper class provides the following method definition to close the connection with the database:

```
public void closeConnection() throws Exception {
    try {
        System.err.println("Closing connection");
        con.close();
    } catch(Exception e) {
        System.err.println
```

```

        ("System Exception in  closeConnection");
        System.err.println(e);
        throw e;
    }
}

```

The complete code listing for the DBHelper class can be found in Appendix C.

4. CORBA Server

The CORBA server forms the broker between the DBHelper class that holds a connection to the database and the client interface located in the web browser. The CORBA server classes expose two IDL interfaces to the client. These are the “NavSecDispenser” and “NavSec” interfaces. The following classes define the functionality of the CORBA server component:

- **NavSecServer.java:** This class acts as the driver for the server classes. NavSecServer accomplishes the following tasks:

Initializes the ORB with arguments passed from the command line:

```

// Initialize the ORB
org.omg.CORBA.ORB orb =
    org.omg.CORBA.ORB.init(args, props);

```

Creates a new NavSecDispenserImpl object and passes it the number of NavSecImpl objects that it should create as a parameter:

```

// Create the NavSecDispenser object
NavSecDispenserImpl mydispenser = new
    NavSecDispenserImpl(args, "My Dispenser",
        noOfInstances);

```

Exports the NavSecDispenserImpl object to the ORB:

```

// Export the newly created object
orb.connect(mydispenser);

```

Gets a reference to the Naming Service and binds the NavSecDispenserImpl object:

```
// Get a reference to the Naming service
org.omg.CORBA.Object nameServiceObj =
    orb.resolve_initial_references ("NameService");

if (nameServiceObj == null) {

    System.out.println ("nameServiceObj = null");
    return;
}

org.omg.CosNaming.NamingContext nameService =
    org.omg.CosNaming.NamingContextHelper.
        narrow(nameServiceObj);

if (nameService == null) {

    System.out.println ("nameService = null");
    return;
}

// bind the NavSecDispenserImpl object in the Naming
// service
NameComponent[] mydispensername = {new
    NameComponent ("NavSecDispenser", "")};

nameService.rebind(mydispensername, mydispenser);
```

The rebind method of the Naming Service is used to replace any previous bindings to the object.

Waits until the program is shut down manually:

```
// wait forever for current thread to die
Thread.currentThread().join();
```

- **NavSecDispenserImpl.java:** NavSecDispenserImpl is a factory class. This class instantiates the pool of NavSecImpl objects. It implements the methods defined in the NavSecDispenser interface, which is defined in the Navy.idl

file. These methods are advertised to clients that will be using them to acquire a reference to a NavSecImpl server object :

```
Interface NavSecDispenser {

    NavSec reserveNavSecObject ()
        raises (navsecException);
    void releaseNavSecObject(in NavSec
                            navsecObject)
        raises (navsecException);
};
```

The constructor for the class gets a reference to the ORB, then instantiates a pool of NavSecImpl objects and exports them to the ORB. The number of NavSecImpl objects to be instantiated is passed to the constructor as an argument. The following is the code for the constructor of the class:

```
public NavSecDispenserImpl(java.lang.String[]
                           args,
                           java.lang.String name,
                           int number) {

    // Register the name with the ORB
    super(name);

    try{

        // get reference to orb
        org.omg.CORBA.ORB orb =
            org.omg.CORBA.ORB.init(args, null);

        // prestart n NavSecImpl Objects
        numObjects = number;

        for (int ix = 0; ix < numObjects; ix++){
            navSec[ix] = new NavSecStatus();
            navSec[ix].ref =
                new NavSecImpl("NavSec" + (ix+1));
            orb.connect(navSec[ix].ref);
            System.out.println
                ("NavSec"+ (ix+1)+ "is ready");
        }
    }
```

```

    } catch(Exception e){
        System.err.println(e);
    }
}

```

NavSecDispenserImpl maintains references to these objects in an array of NavSecStatus objects. When a client asks for the reserveNavSecObject() service, the implementation finds a NavSecImpl object that is available by checking the "inUse" flag in the NavSecStatus object and returns a reference to this object to the client via the ORB. If there are no available NavSecImpl objects a null value is returned. The following code shows how this is accomplished:

```

public Navy.NavSec reserveNavSecObject( )
    throws Navy.navsecException{

    // Find an available NavSecImpl object and
    // return a reference
    for (int ix = 0; ix < numObjects; ix++) {

        if (!navSec[ix].inUse){
            navSec[ix].inUse = true;
            System.out.println
                ("NavSec" + (ix+1) + " reserved.");
            return navSec[ix].ref;
        }
    }

    return null;
}

```

When a client is finished executing any queries, it releases the NavSecImpl object by calling the IDL advertised releaseNavSecObject() method on the NavSecDispenserImpl object. The NavSecDispenserImpl object returns the reference to the pool of available objects by modifying the objects' inUse flag. The following code shows how this is accomplished:

```

public void releaseNavSecObject (Navy.NavSec
                                navsecObject)
                                throws Navy.navsecException {

    // Go through the array of NavSecImpl objects
    // find the right one and change flag to not
    // busy

    for (int ix = 0; ix < numObjects; ix++){
        org.omg.CORBA.Object element =
            (org.omg.CORBA.Object) navSec[ix].ref;
        org.omg.CORBA.Object inelement =
            (org.omg.CORBA.Object) navsecObject;

        if (element._is_equivalent(inelement)){
            navSec[ix].inUse = false;
            System.out.println
                ("NavSec" + (ix+1) + " released.");
            return;
        }
    }

    System.out.println("Reserved Object not found");
    return;

}

```

- **NavSecStatus.java:** This class maintains a handle to NavSec objects by pairing a reference to NavSecImpl object with a Boolean value indicating whether the object is in use or available. The following is the definition for this convenience class:

```

class NavSecStatus{
    NavSecImpl ref;
    boolean inUse;
}

```

```

        NavSecStatus() {
            ref = null;
            inUse = false;
        }
    }
}

```

- **NavSecImpl.java:** This class implements the NavSec interface and is therefore responsible for providing implementations for the services advertised in this interface. Clients that execute queries on the NSGDB database instance use these services. Each NavSecImpl object instantiates a DBHelper object when it is created. The DBHelper class provides JDBC functionality required to execute queries and return results to the calling client. The results are returned as a Vector by the DBHelper object, which is copied into a CORBA struct of the proper size required by this class. An instance of this class is constructed by calling the following constructor:

```

public NavSecImpl(java.lang.String name) {
    // Register the name with the ORB
    super(name);
    try {

        myDBHelper = new DBHelper();
        myDBHelper.connect
            ("www.dbase-research.cs.nps.navy.mil",
             "NSGDB", "sa", "");

        System.out.println
            ("DBHelper Object " + name + " Created");
        instanceName = name;
    } catch (Exception e) {
        System.out.println("System Exception ");
    }
}

```

The name passed to the constructor is used to register an instance of the class to the ORB implementation. The NavSecImpl answers the requests of the clients by making method calls on the DBHelper object to which it holds a reference. DBHelper executes the proper SQL query by using the JDBC classes. The results of the query are then returned to the NavSecImpl object as a Java Vector object. NavSecImpl then translates the results into a CORBA sequence of structs, so that they can be sent back to the client using

the ORB. The following is the definition of such a method used by the CORBA client to get a sailor's leave data stored in the DBMS:

```
public Navy.leavedescStruct[] getLeaveData
    ( java.lang.String sailorSsn,
      java.lang.String lastName ) {
    try {
        Navy.leavedescStruct[] leaveData;
        Vector leaves =
            myDBHelper.getLeaveData
                        (sailorSsn,lastName);
        leaveData =
            new Navy.leavedescStruct[leaves.size()];
        leaves.copyInto(leaveData);
        return(leaveData);
    } catch (Exception e) {
        System.out.println
            ("System Exception in getLeaveData");
        return null;
    }
}
```

There are 40 methods in our EIS prototype that can be used by the client to insert, modify or query database data. The NavSecImpl class provides an implementation for each of these IDL advertised methods by calling proper DBHelper class methods. Complete source code for the NavSecImpl class can be found in Appendix C.

5. Applet Client

The client side of the Naval Security Group EIS is comprised of 85 separate classes. The base class that manages all of the other classes is named Gui.java. This class extends the JApplet class in the java.swing package, which in turn extends the Applet class in the java.awt package. The inheritance and composition relationships of

the Gui class are shown in Figure 7.9. Each of the panels that belong to the GUI class extends the JPanel class in the java.swing package, which extends the java.awt.Component class. The inheritance relation for the JPanel class is not shown to keep the diagram simple.

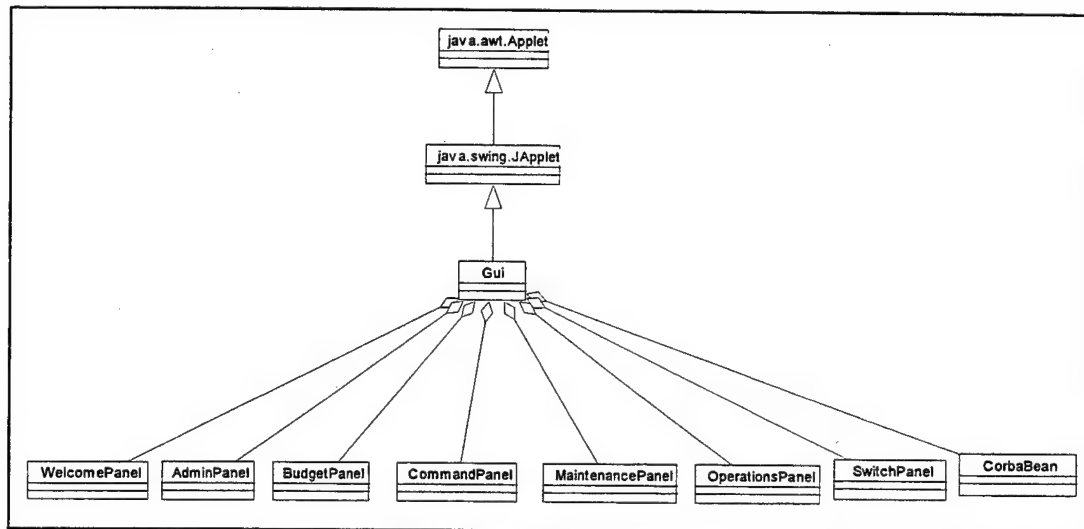


Figure 7.9: GUI class diagram

The Panel classes are themselves comprised of the panels that allow the user to enter, modify or view information maintained in the back-end database. Figure 7.10 shows the class diagram for the AdminPanel class.

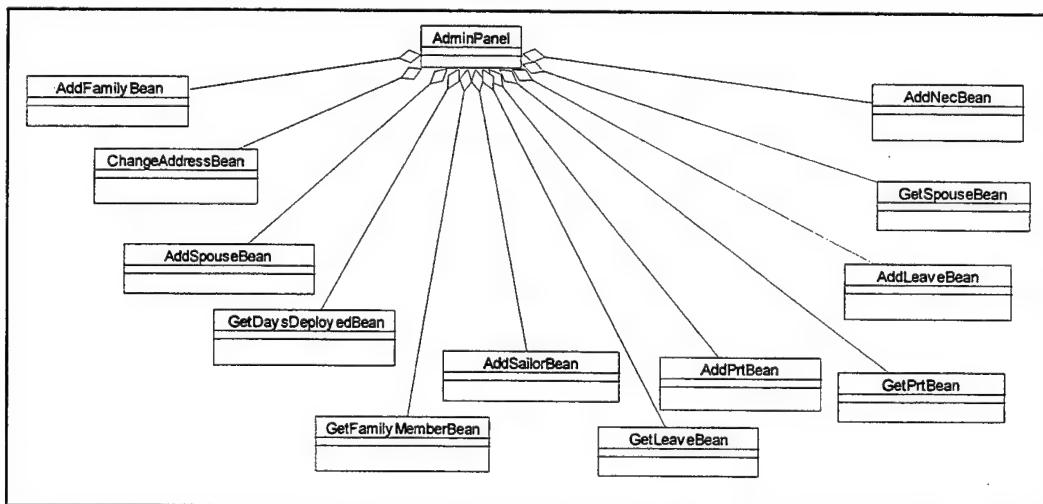


Figure 7.10: AdminPanel class diagram

Each GetXXXXBean class contains two classes that extend java.swing.JPanel. The first JPanel is used to specify the input parameters for a query to be executed on the

database, while the second is used to display the results of the query to the client. We will go over the basic components that make up the client interface in the following sections.

- **Gui.java:** The GUI class holds a reference to six panels that enable the client to use all of the services offered by the EIS. When the browser creates an instance of the GUI class, it creates an instance of the CorbaBean class and passes a reference to itself. The following code shows how this is implemented:

```
CorbaBean corbaBean = new CorbaBean(this);
```

The instance of the CorbaBean class is visible to the entire GUI class, so all of the widgets that make up the interface have access to its methods. The other objects will use the methods of the CorbaBean object to invoke the IDL advertised methods of the server.

When the browser initializes the applet, a panel is displayed for the client to enter a username and password. If the client is authorized he will then be able to access the panels that display the operations that can be performed by the client. The client will interact with these panels to input, modify or view information stored in the back-end DBMS. When the client leaves the HTML page that hosts the GUI applet, the browser will invoke the stop() method on the GUI object. The corbaBean object will be notified to release the connection with the CORBA server in the stop() method of the GUI object as shown in the following code:

```
public void stop() {  
    try{  
        corbaBean.releaseObject();  
        System.err.println("Released the orb.");  
    } catch(Exception e){  
        System.err.println("Could not release the  
                                orb.");  
    }  
}
```

- **CorbaBean.java:** This is a helper class that will be used to invoke methods of the CORBA server that the client interacts with. The CorbaBean object needs a reference to the CORBA server object in order to execute the IDL advertised methods of the server object. The first task that is performed by the CorbaServer object is to initialize the ORB with the following lines of code:

```
//initialize the orb
org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init
                        (parent, null);
```

Applets have a different pattern of initializing the ORB. A reference to the applet must be passed to the init() method of the ORB class. We passed a reference to the GUI class in the code above. The next step that should be accomplished by the CorbaBean object is to get a reference to the Naming Service:

```
// Get a reference to the Naming service
org.omg.CORBA.Object nameServiceObj =
    orb.resolve_initial_references("NameService");

if (nameServiceObj == null) {
    System.out.println("nameServiceObj = null");
    return;
}

org.omg.CosNaming.NamingContext nameService =
    org.omg.CosNaming.NamingContextHelper.
        narrow(nameServiceObj);

if (nameService == null) {
    System.out.println("nameService = null");
    return;
}
```

Then the name of the server object must be created. With the following line of code, we indicate the name of the server object that we want to obtain a reference to as "NavSecDispenser" and the root context as null:

```
org.omg.CosNaming.NameComponent[] mydispensername =  
    {new org.omg.CosNaming.NameComponent  
        ("NavSecDispenser", "")};
```

The next step is asking the NavSecDispenserHelper object, which holds the stub files for the client, to return a reference to the CORBA server object with the indicated name:

```
myDispenser =  
    Navy.NavSecDispenserHelper.narrow  
        (nameService.resolve(mydispensername));
```

The NavSecDispenser is an IDL advertised CORBA factory object that can be used by the client to obtain a reference to the NavSec server object. The NavSec server object provides an implementation of the IDL defined server methods that may be invoked by the client. A reference to the NavSec object is obtained by calling the reserveNavSec() object of the NavSecDispenser object:

```
myNavSec= myDispenser.reserveNavSecObject();
```

The CorbaBean object maintains this reference as a class variable and invokes the proper methods on the NavSec object when requested. The method defined below is used by panel objects to change the data stored in the DBMS regarding a command division:

```
public synchronized boolean changeDepartment  
    (Navy.departmentStruct myDepartmentStruct) {  
    boolean result;  
    try{
```

```

        result = myNavSec.
        changeDeptInfo(myDepartmentStruct);
        System.out.println
            ("Query run successfully");
        return result;
    } catch(Exception e){
        System.out.println
            ("Error in processing query");
        return false;
    }
}

```

If an exception occurs while performing the operation, false value is returned. The calling panel will use this value to display an error message to the client. These methods are declared “synchronized” in order to provide atomicity while performing the operation.

- **Panel classes:** The GUI class maintains references to six classes that extend the java.swing.JPanel class and have similar functionality to each other. These are WelcomePanel, AdminPanel, BudgetPanel, CommandPanel, OperationsPanel and MaintenancePanel. Each of these panel objects contains a number of bean objects that can be used by the client to perform a specific operation. We will examine the AdminPanel class as an example. AdminPanel holds all of the panels that a client requires for the insertion, modification or viewing of Administration related data in the database. These panels are in the form of bean classes, which extend the java.swing.JPanel class. Figure 7.10 shows these classes. They are presented to the client in a tabular layout. When the client clicks on a tab, the proper bean object will be displayed. The client can then interact with the bean object to perform operations on the back-end database.

- **Bean classes:** Each bean class is comprised of one or two panels. The number of panels that each bean class contains depends on the operation that it is designed to perform. The bean classes that are used by the client to insert information into the database have one panel to get user inputs. Other bean classes used by the client to execute select queries and view the resulting information have two panels. One of these panels is used to enter the parameters for the query, the other is used to display the results returned via the ORB. All of the bean classes make calls on the CorbaBean object to execute the operations. After the client enters the parameters for the query and clicks on the "Submit" button the generated event is captured by the bean, the input data is parsed and the corresponding method of the CorbaBean object is invoked with the input data as parameters. The following code shows how the GetDaysDeployedBean object in the AdminPanel invokes the CorbaBean method to get information about the deployment days of a sailor:

```
Navy.deploymentStruct[] result =  
    parentApplet.corbaBean.getDaysDeployed  
        (entryString[0],entryString[1],  
         entryString[2],entryString[3]);
```

The results of the query are returned in an array of CORBA structs. If an error condition occurs while performing the query, or the result set has no elements, this is indicated to the user in a dialog box displayed by the calling panel object. This is accomplished with the following code:

```
if( result == null ){  
    JOptionPane.showMessageDialog  
        (parentApplet,"Error in executing query"  
         , "", JOptionPane.ERROR_MESSAGE );  
}  
else{  
    int resultLength = result.length;
```

```

        if( resultLength == 0 ){
            JOptionPane.showMessageDialog
                (parentApplet,"The query returned no results"
                , "", JOptionPane.INFORMATION_MESSAGE);
        }
    }
}

```

After making these checks the results of the query are parsed and displayed by using an instance of the `java.swing.JTable` class. This class is useful for displaying tables of information.

F. SYSTEM OPERATION SCENARIO

After installing and configuring Java and Visibroker ORB on the machine that will host the CORBA server, the following steps should be followed to provide services to clients:

- First, start the Visibroker OSAgent. The Naming Service uses OSAgent transparently. The OSAgent may be started automatically as a system service or by typing the following command:

```
prompt> start osagent - c
```

The start command opens up a new window for execution so you can keep using the same window for the following commands.

- Start the CORBA Naming Service by entering the following command:

```

prompt> java -DORBServices=CosNaming
        -DSVCnameroot=CorbaJava
        -DJDKrenameBug
        com.visigenic.vbroker.services.
            CosNaming.ExtFactory
        JavaCorba namingLog

```


- Then we can start the server by entering the following:

```
prompt> java -DORBServices=CosNaming  
          -DSVCnameroot=JavaCorba  
          NavSecServer
```

In order to provide an Internet accessible interface for the client, the following steps need to be taken:

- Install the HTML files and the Java applet classes on the web server. The machine that hosts the web server must have access to the ORB classes. This can be achieved by configuring the CLASSPATH environment variable to point to the “Visibroker\lib” directory or unzipping the jar files found in this directory to the directory that contains client HTML and Java files.
- Start the IIOP Gatekeeper. This enables the applet to invoke CORBA servers that are on a different host than the web server. It also allows firewall traversal. This is accomplished by entering the following:

```
prompt> start gatekeeper
```

- Enter the URL for the CORBA server in your web browser. For our EIS prototype the URL is <http://www.dbase-research.nps.navy.mil:15000/Gui.html>.

- Enter your username and password. Figure 7.11 displays a screenshot of the log in process.

A screenshot of a web-based login interface. At the top left, the word "MENU" is visible. The main content area has a light gray background. In the center, there is a rectangular box containing the text "WELCOME TO NAVY SECURITY GROUP ENTERPRISE INFORMATION SYSTEM". Below this box, there are two input fields: the first is labeled "LOG ON NAME" and the second is labeled "PASSWORD". Below the password field is a button labeled "ENTER".

Figure 7.11: Prototype log-in screenshot

- Select the menu item that you wish to view. Figure 7.12 displays the applet switchboard.

A screenshot of a web-based applet switchboard. At the top left, the word "MENU" is visible. The main content area has a light gray background. In the center, there is a rectangular box containing the text "NAVY SECURITY GROUP EIS". Below this box, there are six rectangular buttons arranged in two columns. The left column contains three buttons labeled "ADMINISTRATION", "COMMAND", and "OPERATIONS". The right column contains three buttons labeled "BUDGET & SUPPLY", "MAINTENANCE", and "WELCOME".

Figure 7.12: Applet switchboard

- Execute the queries that you want to perform. Figure 7.13 displays an example of a database query.

NAVY SECURITY GROUP EIS COMMAND

View Command Demographics By Nec View Command Demographics By Rank View Command Emergency List

Add Department Add Division Change Department Data Change Division Data View Command Demographics By Gender

GET EMERGENCY LIST FOR THE COMMAND FORM

COMMAND ID

Enter the command id and click on the submit button

BACK CLEAR SUBMIT CANCEL

Figure 7.13: Sample prototype database query screenshot

- View the results of the query. Figure 7.14 displays an example of a query result set.

NAVY SECURITY GROUP EIS COMMAND

View Command Demographics By Nec View Command Demographics By Rank View Command Emergency List

Add Department Add Division Change Department Data Change Division Data View Command Demographics By Gender

GET EMERGENCY LIST FOR THE COMMAND FORM

Command Name	Department Id	Rank	First Name	Last Name	Home Phone
NSGA Alpha	00	CAPT	William	White	3447659982
NSGA Alpha	00	CTACM	Daniel	Dante	7589398501
NSGA Alpha	01	CDR	Joan	Blue	7788830458
NSGA Alpha	10	CTACS	Gregory	Elm	7859803480
NSGA Alpha	10	LCDR	Ronald	Rose	7873947676
NSGA Alpha	30	CTMC	Frank	Ford	4723987593
NSGA Alpha	30	LCDR	Dimitri	Orange	8748885999
NSGA Alpha	50	CTRC	Cassandra	Granite	9843983759
NSGA Alpha	50	LCDR	Elaine	Violet	7883499549

BACK CLEAR SUBMIT CANCEL

Figure 7.14: Sample prototype query result screenshot

- Exit the web page or shut down the web browser when you are finished.

VIII. CONCLUSION

A. SYNOPSIS AND CONCLUSION

With the ending of the cold war, military organizations in the United States faced considerable cuts in personnel, material and financial resources. All remaining organizations were placed under considerable pressure to do more with less. The ever-changing nature of organizations placed the Naval Security Group in a situation where it had to manage and integrate widely dispersed information stores that resided on different hardware and used different software. The sharing and synchronization of such dispersed data has become a burden on the NSG. Many similar organizations in the military and corporate America are adapting distributed object technologies that provide them with a manageable, efficient and cost-effective information infrastructure. These systems are proven to be scalable, high performing, and easy to implement, manage and update.

The focus of this thesis has been to design and implement a component-based Client/Server system that makes use of the Internet as the network medium. Component-based distributed systems offer lower costs and shorter development cycles. They integrate with existing software, which enables an organization to protect its investments. The non-proprietary nature of CORBA was a major factor in choosing this new technology for our proposed solution. We believe that locking an organization into a one-vendor solution will greatly limit that organization's information technology capabilities. A distributed application built using CORBA gives the user hardware and operating system independence. Organizations can mix and match systems according to their needs and features supported by these products. The backbone of our EIS, the Internet, has become the most widely used medium for information transfer. Its almost non-existent cost of maintenance, high availability and usability make it the preferred method for implementing distributed architectures.

This thesis has proven that it is an effective solution to implement an Internet-based Enterprise Information System using Commercial Off The Shelf tools. Our proposed system uses CORBA, an industry-backed, non-proprietary, standards-based distributed architecture and Java, a new object-oriented high-level language that enables

developers to deliver mobile executable content over the Internet. We have found that a combination of these technologies provides a powerful infrastructure for building component-based Client/Server systems.

This thesis can be used by the Naval Security Group to evaluate a number of similar commercial products that have begun to emerge, or it can be used to further develop our prototype into a fully functional EIS. In each case, the end result would enhance the capabilities of the organization with very little overhead involved.

B. SECURITY

Any system that is connected to the Internet is susceptible to attacks. These attacks can harm the confidentiality, integrity and availability of information. Placing a firewall between the Internet and the CORBA server can serve as a first line of defense for our proposed EIS. The firewall can allow only packets that use specific ports on the server machine to pass through. These packets need to be addressed to the Gatekeeper, which acts as a proxy between the server implementation and the client applet. A buffer zone can be created with the addition of yet another firewall between the Gatekeeper and the CORBA server. Such a configuration would make it even harder for an intruder to break in.

Even if all of the servers are protected from intrusion by the use of firewalls, it is impossible to ensure the confidentiality of packets that traverse the Internet without the use of an encryption package. Since the Internet was designed as an open medium with no security considerations, it is possible to view the contents of any packet with use of a simple packet sniffer program. The OMG provides a standard for using IIOP over the Secure Socket Layer (SSL) protocol. SSL allows users of the system to encrypt the information before it is sent over the Internet. It also supplies the means to identify and authenticate each side of a data exchange. This is extremely important, as it is possible to fake one's identity utilizing IP spoofing methods. OMG certificates authenticate each user in the SSL system. Each certificate contains the name of the holder and a value. The certificates must be verified by a trustworthy authority in order to be considered authentic. There are various firms that specialize in the generation and distribution of authentic certificates. When the client and the server establish a connection over a SSL

they exchange their certificates. The certificates can then be used to verify the identity of the party in an exchange. In order to use the Visibroker SSL, both the client and the server must use SSL Version 3.0. If these capabilities are not considered secure enough for the EIS, the system can be implemented on a more secure Intranet.

C. AREAS FOR FURTHER RESEARCH

In our implementation we used a basic object dispenser that depends on the server platform for scaling and load balancing. In a real world solution a Transaction Processing (TP) monitor would be required to handle all of the server objects and connections with clients. A commercial off the shelf TP monitor could be purchased to provide scalability to the EIS. The Visibroker SSL package can also be purchased to provide security for Internet-based communications. Since our major research question did not cover the usability of the interface, the user interface has been designed only to provide basic functionality. It can be further enhanced to provide error detection and correction and the functionality can be increased through the use of different widgets that can be found in the java.swing package or different GUI builder tools such as Jbuilder2 and VisualCafe3.0. A GUI-based tool can be developed to provide the server manager with the capability to monitor the system from any location over the Internet by the use of a web browser. Different implementations that make use of the competing component technologies, such as Remote Method Invocation (RMI) and Distributed Component Object Model (DCOM), can be built and benchmarked to show a comparison of the performance and scalability of these different distributed architectures. The use of applets places severe limitations on the capabilities of the user interface. For example, the user cannot store the results of a query on his hard disk. These limitations can be relaxed with the use of the new security model in JDK1.1, which uses certificates to authenticate the source of the applet and then grants permissions to the applet in accordance with the user defined security policy on the client system. JDK1.2 will enhance the granularity of this security model. An implementation with JDK1.2 and CORBA 3.0 can be developed to investigate such enhancements. Finally, the system front-end can be built as an application instead of an applet. This would mean sacrificing the automatic upgrade feature of the system, but with the installation of the ORB locally



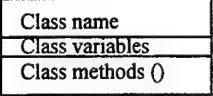
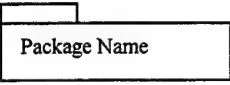
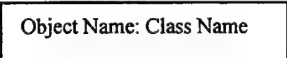
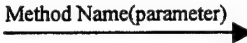
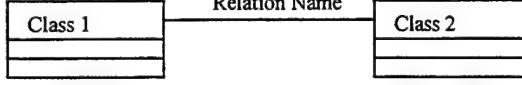

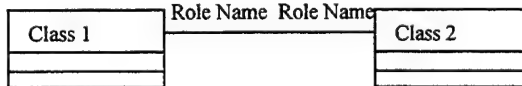

on the client, the penalty of having to download the ORB everytime the user needs access to the system would be avoided. The use of callbacks by the CORBA server objects enables updating the state of the applet front-end when the data queried by the applet changes after the query has been executed. This brings rapid response to the changes that occur almost every second. For example, the user may run a query to determine the number of missions flown in a specific area. There might be two more missions added to the database by the time the user has observed the query results. By using callbacks the CORBA server can update the results of the query while the user is viewing them. The EIS model implemented in this thesis can be expanded to cover and evaluate the use of callbacks by the server objects. Finally, implementations that make use of Message Oriented Middleware (MOM) can be built for evaluation. Message oriented middleware enable asynchronous messages to be exchanged between the server and the client. They can be used in environments where the client and server will not be operable all the time. For example the client can be hosted by a mobile unit that has sporadic access to the EIS. This will expand the usable environment of the EIS from stationery locations to mobile units with an Internet connection.


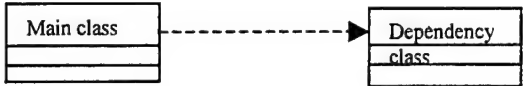
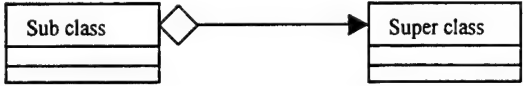

LIST OF REFERENCES

1. Stevenson, J., *An Enterprise Information System For The Naval Security Group*, Master's Thesis, Naval Postgraduate School, Monterey, California, March 1998.
2. MTIAC Publications, *Executive Information Systems*,
<http://mtiac.hq.itri.com/MTIAC/pubs/eis/eis1.htm>.
3. Orfali, R., Harkey, D., *Client Server Programming with Java and CORBA*, Wiley, 1996.
4. JavaSoft, *JDK 1.1 JDBC Package Documentation*,
<http://www.java.sun.com/products/jdk/1.1/docs/guide/jdbc>.
5. Fedorov, A., Harrison, R., et al., *Professional Active Server Pages*, Second Edition, Wrox Press Inc, 1998.
6. Orfali, R., Harkey, D., *The Essential Client/Server Survival Guide*, Wiley, 1996.
7. Deitel, H. M., Deitel, P.J., *Java How To Program*, Second Edition, Prentice Hall, 1998.
8. Hamilton, G., Cattell, R., Fisher, M., *JDBC Database Access with Java*, Addison-Wesley, 1997.
9. JavaSoft, *JDK 1.1 API Documentation*,
<http://www.javasoft.com/products/jdk/1.1/docs/api/java.sql.Connection.html>
10. Mowbray, T. J., Ruh, W. A., *Inside CORBA*, Addison Wesley, 1997.

11. Pedrick, D., Weedon, J., Goldberg, J., Bleifield, E., *Programming with Visibroker*, Wiley, 1998.
12. Rosenberger, J. R., *Teach Yourself CORBA in 14 Days*, Sams Publishing, 1998.
13. Inprise, *Distributed Object Computing In The Internet Age*, White Paper, <http://www.inprise.com/Visibroker/papers/distributed/wp.html>.
14. Vogel, A., Duddy, K., *Java Programming with CORBA*, Wiley, 1998.

APPENDIX A. UML NOTATION REFERENCE TABLE

REPRESENTATION IN DIAGRAM	EXPLANATION
	UML Notation for an actor
	UML Notation for a use case
	UML Notation for a class
	UML Notation for a package
	UML Notation for an object
	UML Notation for a message between objects
	UML Notation for an association relationship
	UML Notation for an aggregation relationship
	UML Notation for displaying role names
	UML Notation for displaying a multiplicity indicators

 <p>The diagram shows two class boxes. The left box is labeled 'Sub class' and the right box is labeled 'Super class'. A solid line with an open arrowhead points from the 'Sub class' box to the 'Super class' box, representing an inheritance relationship.</p>	<p>UML Notation for displaying a inheritance relationship</p>
 <p>The diagram shows two class boxes. The left box is labeled 'Main class' and the right box is labeled 'Dependency class'. A dashed line with an open arrowhead points from the 'Main class' box to the 'Dependency class' box, representing a dependency relationship.</p>	<p>UML Notation for displaying a dependency relationship</p>
 <p>The diagram shows two class boxes. The left box is labeled 'Sub class' and the right box is labeled 'Super class'. A solid line with an open diamond at the 'Sub class' end and an open arrowhead at the 'Super class' end points from the 'Sub class' box to the 'Super class' box, representing a unidirectional association relationship.</p>	<p>UML Notation for displaying a unidirectional association relationship</p>
 <p>The diagram shows two package boxes. Each box has a small tab at the top left. A dashed line with an open arrowhead points from the left package box to the right package box, representing a package relationship.</p>	<p>UML Notation for package relationships</p>

APPENDIX B. NAVSEC.IDL FILE

This file is used by the idl2java compiler to create the server skeleton and client stub files. IDL files act as a contract between the server and potential clients.

```
//-----  
// Filename      : NavSec.idl  
// Authors       : Murat Akbay & Steve Lewis  
// Date          : 10/13/1998  
// Compiler      : JDK1.1.6 with Symantec JIT Compiler  
//-----  
  
module Navy  
{  
    exception navsecException  
    {string reason;  
    };  
  
    struct adressStruct  
    {  
        string address;  
        string city;  
        string state;  
        string country;  
        string postalCode;  
        string homePhone;  
        string ssn;  
    };  
    typedef sequence<adressStruct> adressSeq;  
  
    struct familyStruct  
    {  
        string ssn;  
        string name;  
        string sdate;  
        string sex;  
        string sailorSsn;  
    };  
    typedef sequence<familyStruct> familySeq;  
  
    struct familydescStruct  
    {  
        string rank;  
        string firstName;  
        string lastName;  
        string memberSsn;  
        string memberName;  
        string memberBirth;  
        string memberSex;
```

```

    string ememberSailorSsn;
};
typedef sequence<familydescStruct> familydescSeq;

struct spouseStruct
{
    string ssn;
    string name;
    string bdate;
    string gender;
    string sailorSsn;
    string update;
};
typedef sequence<spouseStruct> spouseSeq;

struct spousedescStruct
{
    string rank;
    string firstName;
    string lastName;
    string spouseSsn;
    string spouseName;
    string spouseBirth;
    string spouseSex;
    string SailorSsn;
};
typedef sequence<spousedescStruct> spousedescSeq;

struct prtStruct
{
    string commandId;
    string identifier;
    string height;
    string weight;
    string pushUps;
    string sitUps;
    string runTime;
    string pdate;
    string status;
    string ssn;
};
typedef sequence<prtStruct> prtSeq;

struct prtdescStruct
{
    string rank;
    string firstName;

```

```

    string lastName;
    string commandId;
    string prtId;
    string prtSsn;
    string height;
    string weight;
    string pushUps;
    string sitUps;
    string runTime;
    string prtDate;
    string status;
};
typedef sequence<prtdescStruct> prtdescSeq;

```

```

struct leaveStruct
{
    string commandId;
    string leaveControlNumber;
    string typeOfLeave;
    string departDate;
    string returnDate;
    string ssn;
};
typedef sequence<leaveStruct> leaveSeq;

```

```

struct leavedescStruct.
{
    string rank;
    string firstName;
    string lastName;
    string leaveSsn;
    string controlNumber;
    string typeOfLeave;
    string dateDepart;
    string dateReturn;
};
typedef sequence<leavedescStruct> leavedescSeq;

```

```

struct necStruct
{
    string primaryNec;
    string secondaryNec;
    string tertiaryNec;
    string lastDate;
    string ssn;
};
typedef sequence<necStruct> necSeq;

```

```

struct propertyStruct
{
    string propertyId;
    string propertyDesc;
    string purchaseDate;
    string price;
    string commandId;
    string purchaser;
    string sqlDate;
    string purchaseCategory;
};
typedef sequence<propertyStruct> propertySeq;

struct departmentStruct
{
    string commandId;
    string departmentId;
    string title;
    string headSsn;
    string chiefSsn;
    string telephone;
    string email;
};
typedef sequence<departmentStruct> departmentSeq;

struct divisionStruct
{
    string commandId;
    string departmentId;
    string divisionId;
    string title;
    string headSsn;
    string chiefSsn;
    string telephone;
    string email;
};
typedef sequence<divisionStruct> divisionSeq;

struct maintenanceStruct
{
    string person;
    string maintenanceNumber;
    string itemId;
    string pmsNumber;
    string dateOfConduct;
    string priority;
    string emDescription;
    string commandId;
    string departmentId;

```



```

    string divisionId;
    string hoursOnJob;
};
typedef sequence<maintenanceStruct> maintenanceSeq;
struct sailorStruct
{
    string ssn;
    string firstName;
    string middleName;
    string lastName;
    string sex;
    string rankRate;
    string dateOfRank;
    string device1;
    string device2;
    string device3;
    string primaryNec;
    string secondaryNec;
    string tertiaryNec;
    string address;
    string city;
    string state;
    string country;
    string postalCode;
    string homePhone;
    string dateOfBirth;
    string dateOfService;
    string homeTown;
    string homeState;
    string dateOfUpdate;
};
typedef sequence<sailorStruct> sailorSeq;

```

```

struct seamissionStruct
{
    string missionNumber;
    string dateOfMission;
    string typeOfMission;
    string exerciseName;
    string platformMode;
    string platformType;
    string platformNumber;
    string platformName;
    string missionArea;
};
typedef sequence<seamissionStruct> seamissionSeq;

```

```

struct airmissionStruct
{
    string missionNumber;

```

```

    string takeOffTime;
    string landingTime;
    string duration;
    string platformNumber;
    string squadron;
    string squadronCrew;
    string base;
    string city;
    string country;
    string typeMission;
    string platformType;
    string exerciseName;
    string missionArea;
    string trackId;
};
typedef sequence<airmissionStruct> airmissionSeq;

struct deploymentStruct
{
    string tangoNumber;
    string dateDepart;
    string rankRate;
    string firstName;
    string lastName;
    string commandName;
    string title;
    string dateArrival;
    string dateDeparture;
    string comm;
};
typedef sequence<deploymentStruct> deploymentSeq;

struct propertydescStruct
{
    string commandName;
    string propertyId;
    string propertyDesc;
    string dateOfPurchase;
    string price;
    string rank;
    string firstName;
    string lastName;
};
typedef sequence<propertydescStruct> propertydescSeq;

struct demographicStruct
{
    string commandName;
    string number;

```

```

    string datex;
};
typedef sequence<demographicStruct> demographicSeq;

struct emergencyStruct
{
    string commandName;
    string deptId;
    string rank;
    string firstName;
    string lastName;
    string homePhone;
};
typedef sequence<emergencyStruct> emergencySeq;

struct seamissiondescStruct
{
    string missionNumber;
    string dateOfMission;
    string typeOfMission;
    string exerName;
    string platformMode;
    string platformType;
    string platformNumber;
    string platformName;
    string missionAreaId;
    string description;
};
typedef sequence<seamissiondescStruct> seamissiondescSeq;

struct airmissiondescStruct
{
    string missionNumber;
    string dateTakeOff;
    string dateLanding;
    string duration;
    string platformType;
    string platformNumber;
    string squadronId;
    string squadronCrew;
    string stageBase;
    string stageCity;
    string stageCountry;
    string typeOfMission;
    string exerName;
    string missionAreaId;
    string trackId;
    string description;
};

```

```
typedef sequence<airmissiondescStruct> airmissiondescSeq;
```

```
struct commaintenanceStruct
```

```
{
    string commandId;
    string description;
    string person;
    string itemId;
    string maintNo;
    string dateConduct;
    string priority;
    string emergencyDesc;
    string deptId;
    string hours;
};
```

```
typedef sequence<commaintenanceStruct> commaintenanceSeq;
```

```
struct itemmaintenanceStruct
```

```
{
    string pmsDesc;
    string description;
    string person;
    string itemId;
    string maintNo;
    string dateConduct;
    string priority;
    string emergencyDesc;
    string deptId;
    string hours;
};
```

```
typedef sequence<itemmaintenanceStruct> itemmaintenanceSeq;
```

```
interface NavSec
```

```
{
    boolean          addAddress(in adressStruct adressData);
    boolean          addFamily(in familyStruct familyData);
    boolean          addSpouse(in spouseStruct spouseData);
    boolean          addPrt(in prtStruct prtData);
    boolean          addLeave(in leaveStruct leaveData);
    boolean          addNec(in necStruct necData);
    boolean          addProperty(in propertyStruct propertyData);
    boolean          addDepartment(in departmentStruct
                                   departmentData);
    boolean          addDivision(in divisionStruct divisionData);
    boolean          addMaintLog(in maintenanceStruct
                                   maintenanceData);
    boolean          addSailor(in sailorStruct sailorData);
    boolean          addSeaMission(in seamissionStruct
                                   seamissionData);
}
```

boolean	addAirMission(in airmissionStruct airmissionData);
boolean	changeDeptInfo(in departmentStruct departmentData);
boolean	changeDivisionInfo(in divisionStruct divisionData);
boolean	getAuthorization(in string userName, in string password);
familydescSeq	getFamilyMembers(in string sailorSsn, in string lastName);
spousedescStruct	getSpouse(in string sailorSsn, in string lastName);
leavedescSeq	getLeaveData(in string sailorSsn, in string lastName);
prtdescSeq	getPrtResults(in string sailorSsn, in string lastName);
deploymentSeq	getDaysDeployed(in string sailorSsn, in string lastName, in string startDate, in string stopDate);
propertydescSeq	getPropertyList(in string commandId);
demographicSeq	getComDemByGender(in string commandId, in string sex);
demographicSeq	getComDemByNec(in string commandId, in string nec1, in string nec2, in string nec3);
demographicSeq	getComDemByRank(in string sailorSsn, in string paygrade);
emergencySeq	getEmergencyList(in string commandId);
seamissiondescSeq	getSeaMissions(in string platform, in string typeMission, in string missionArea, in string startDate, in string stopDate);
seamissiondescSeq	getSeaMissionsByPlatform(in string platform, in string startDate, in string stopDate);
seamissiondescSeq	getSeaMissionsByArea(in string missionArea, in string startDate, in string stopDate);
seamissiondescSeq	getSeaMissionsByType(in string typeMission, in string startDate, in string stopDate);
seamissiondescSeq	getSeaMissionsByExercise(in string exerciseName, in string startDate, in string stopDate);

```

seamissiondescSeq    getSeaMissionsByPlatformName(in string
                                                platform,
                                                in string
                                                startDate,
                                                in string
                                                stopDate);

airmissiondescSeq    getAirMissions(in string squadron,
                                      in string platform,
                                      in string typeMission,
                                      in string missionArea,
                                      in string startDate,
                                      in string stopDate);

airmissiondescSeq    getAirMisByPlatformNo(in string platform,
                                             in string startDate,
                                             in string stopDate);

airmissiondescSeq    getAirMisByArea(in string missionArea,
                                       in string startDate,
                                       in string stopDate);

airmissiondescSeq    getAirMisByType(in string typeOfMission,
                                       in string startDate,
                                       in string stopDate);

airmissiondescSeq    getAirMisByExercise(in string exerciseName,
                                           in string startDate,
                                           in string stopDate);

airmissiondescSeq    getAirMisBySquadron(in string squadronName,
                                           in string startDate,
                                           in string stopDate);

commaintenanceSeq    getMaintByCommand(in string commandId,
                                         in string departmentId,
                                         in string startDate,
                                         in string stopDate);

itemmaintenanceSeq   getMaintByItem(in string itemId);
};

interface NavSecDispenser
{
    NavSec reserveNavSecObject() raises (navsecException);
    void releaseNavSecObject(in NavSec navsecObject)
        raises (navsecException);
};
};

```

APPENDIX C. SERVER FILES

This appendix provides all the source code required for the Server implementation. It includes the NavSecServer.java, NavSecDispenserImpl.java, NavSecImpl.java, and DBHelper.java classes.

```
//-----
// Filename      : NavSecServer.java
// Authors       : Murat Akbay & Steve Lewis
// Date          : 10/13/1998
// Compiler      : JDK1.1.6 with Symantec JIT Compiler
//-----

/**
 * Main driver class for NavSecServer, server component of the NavSec
 * EIS Prototype
 *
 * @authors Murat Akbay & Steve Lewis
 */

import org.omg.CosNaming.*;
import java.util.*;

class NavSecServer{
    static public void main(String[] args){

        int noOfInstances;

        try{
            if (args.length == 0){
                noOfInstances = 3;
            }
            else{
                noOfInstances = Integer.parseInt(args[0]);
            }

            // Initialize the ORB
            org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args, null);

            // Create the ClubMedDispenser object
            NavSecDispenserImpl mydispenser =
                new NavSecDispenserImpl(args, "My Dispenser",
                                         noOfInstances);

            // Export the newly created object
            orb.connect(mydispenser);
        }
    }
}
```

```

        System.out.println("Dispenser created..");

        // Get a reference to the Naming service
        org.omg.CORBA.Object nameServiceObj =
            orb.resolve_initial_references ("NameService");

        if (nameServiceObj == null)
        {
            System.out.println("nameServiceObj = null");
            return;
        }

        org.omg.CosNaming.NamingContext nameService =
            org.omg.CosNaming.NamingContextHelper.narrow(nameServiceObj);

        if (nameService == null)
        {
            System.out.println("nameService = null");
            return;
        }

        // bind the NavSec object in the Naming service
        NameComponent[] mydispensername = {new NameComponent
                                            ("NavSecDispenser", "")};

        nameService.rebind(mydispensername, mydispenser);

        System.out.println("Server is ready ...");

        // wait forever for current thread to die
        Thread.currentThread().join();

    } catch (Exception e) {
        System.err.println(e);
    }
}
} // End NavSecServer.java

```



```

//-----
// Filename      : NavSecDispenserImpl.java
// Author       : Murat Akbay & Steve Lewis
// Date        : 10/13/1998
// Compiler     : JDK1.1.6 with Symantec JIT Compiler
//-----

/**
 * This class is the object factory that creates NavSec objects, the
 * number of NavSec Objects is determined by the number of arguments
 * passed to the constructor
 *
 * @author Murat Akbay & Steve Lewis
 */

public class NavSecDispenserImpl extends Navy._NavSecDispenserImplBase{

    private int maxObjects = 10;
    private int numObjects = 0;
    private NavSecStatus[] navSec = new NavSecStatus[maxObjects];

    /**
     * Constructor for the class
     *
     * @param      args      ORB initialization parameter
     *             name      Name used to define the instance of this
     *                       class
     *             number     The number of NavSec instances desired
     * @return     an instance of the class
     * @exception  none
     */
    public NavSecDispenserImpl(java.lang.String[] args,
                               java.lang.String name, int number){

        super(name);

        try{

            // get reference to orb
            org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args, null);

            // prestart n navSec Objects
            numObjects = number;

            for (int ix = 0; ix < numObjects; ix++){
                navSec[ix] = new NavSecStatus();
                navSec[ix].ref = new NavSecImpl("NavSec" + (ix+1));
                orb.connect(navSec[ix].ref);
                System.out.println("NavSec"+ (ix+1)+ "is ready");
            }

        }
    }
}

```

```

    }
    catch(Exception e){
        System.err.println(e);
    }
}

/**
 * Default constructor for the class
 *
 * @param      none
 * @return     an instance  of the class
 * @exception  none
 */
public NavSecDispenserImpl(){
    super();
}

/**
 * This method is used by the client to reserve NavSec Objects, when
 * the NavSec object is reserved it can not be assigned to any other
 * client
 *
 * @param      none
 * @return     a reference to an instance  of NavSec class
 * @exception  none
 */
public Navy.NavSec reserveNavSecObject( )
    throws Navy.navsecException{

    for (int ix = 0; ix < numObjects; ix++)
    {
        if (!navSec[ix].inUse){
            navSec[ix].inUse = true;
            System.out.println("NavSec" + (ix+1) + " reserved.");
            return navSec[ix].ref;
        }
    }

    return null;
}

```

```

/**
 * This method is used by the client to release a NavSec object
 * when it finishes using the object. The released object can be
 * assigned to another client for use.
 *
 * @param NavSec Reference to the NavSec object that the client uses
 * @return      none
 * @exception   none
 */
public void releaseNavSecObject(Navy.NavSec navsecObject)
    throws Navy.navsecException
{
    org.omg.CORBA.Object inelement =
        (org.omg.CORBA.Object) navsecObject;
    for (int ix = 0; ix < numObjects; ix++){
        org.omg.CORBA.Object element =
            (org.omg.CORBA.Object) navSec[ix].ref;

        if (element._is_equivalent(inelement)){
            navSec[ix].inUse = false;
            System.out.println("NavSec" + (ix+1) + " released.");
            return ;
        }
    }

    System.out.println("Reserved Object not found");
    return ;
}
} // End NavSecDispenserImpl.java

```

```

/**
 * This class is used by the NavSecDispenserImpl to keep
 * track of the status of the NavSec objects.
 *
 * @author Murat Akbay & Steve Lewis
 */
class NavSecStatus{

    NavSecImpl ref;
    boolean inUse;

    NavSecStatus(){
        ref = null;
        inUse = false;
    }
} // End NavSecStatus.java

```

```

//-----
// Filename    : NavSecImpl.java
// Author      : Murat Akbay & Steve Lewis
// Date       : 10/14/1998
// Compiler    : JDK1.1.6 with Symantec JIT Compiler
//-----

/**
 * This class is the server object that answers the requests made by
 * the client. It does this by making method calls on the local
 * DBHelper object
 *
 * @author Murat Akbay & Steve Lewis
 */

import java.util.*;
public class NavSecImpl extends Navy._NavSecImplBase
{
    private DBHelper myDBHelper;
    private String instanceName;

    /**
     * Constructor for the class
     *
     * @param    name    Name used to define the instance of this class
     * @return   none
     * @exception none
     */
    public NavSecImpl(java.lang.String name) {
        super(name);
        try {

            myDBHelper = new DBHelper();
            myDBHelper.connect("131.120.3.4", "NSGDB", "sa", "");
            System.out.println("DBHelper Object " + name + " Created");
            instanceName = name;
        } catch (Exception e){
            System.out.println("System Exception ");
        }
    }

    /**
     * Default constructor for the class
     *
     * @param    none
     * @return   none
     * @exception none
     */
    public NavSecImpl(){
        super();
    }
}

```

```

/**
 * This method is used by the client to modify the address of a
 * sailor in the database
 *
 * @param      addressData The struct that holds the address
 *              information
 * @return      boolean true or false
 * @exception   none
 */
public boolean addAddress(Navy.adressStruct addressData ){
    try {
        return myDBHelper.addAddress(addressData);
    } catch (Exception e){
        System.out.println("System Exception in addAddress");
        return false;
    }
}

/**
 * This method is used by the client to add family members
 * to the database
 *
 * @param      familyData The struct that holds family information
 * @return      boolean true or false
 * @exception   none
 */
public boolean addFamily(Navy.familyStruct familyData ){
    try {
        return myDBHelper.addFamily(familyData);
    } catch (Exception e){
        System.out.println("System Exception in addFamily");
        return false;
    }
}

/**
 * This method is used by the client to add spouse data
 * to the database
 *
 * @param      spouseData The struct that holds spouse information
 * @return      boolean true or false
 * @exception   none
 */
public boolean addSpouse(Navy.spouseStruct spouseData ){
    try {
        return myDBHelper.addSpouses(spouseData);
    } catch (Exception e){
        System.out.println("System Exception in addSpouses");
        return false;
    }
}

```

```

    }

/**
 * This method is used by the client to add PRT data
 * to the database
 *
 * @param      prtData The struct that holds PRT information
 * @return      boolean true or false
 * @exception   none
 */
public boolean addPrt(Navy.prtStruct prtData ){
    try {
        return myDBHelper.addPrt(prtData);
    } catch (Exception e){
        System.out.println("System Exception in addPrt");
        return false;
    }
}

/**
 * This method is used by the client to add leave data
 * to the database
 *
 * @param      leaveData The struct that holds leave information
 * @return      boolean true or false
 * @exception   none
 */
public boolean addLeave(Navy.leaveStruct leaveData ){
    try {
        return myDBHelper.addLeave(leaveData);
    } catch (Exception e){
        System.out.println("System Exception in addLeave");
        return false;
    }
}

/**
 * This method is used by the client to add nec data
 * to the database
 *
 * @param      necData The struct that holds nec information
 * @return      boolean true or false
 * @exception   none
 */
public boolean addNec(Navy.necStruct necData ){
    try {
        return myDBHelper.addNec(necData);
    } catch (Exception e){
        System.out.println("System Exception in addNec");

```

```

        return false;
    }
}

/**
 * This method is used by the client to add command property data
 * to the database
 *
 * @param      propertyData The struct that holds property
 *              information
 * @return     boolean true or false
 * @exception  none
 */
public boolean addProperty(Navy.propertyStruct propertyData ){
    try {
        return myDBHelper.addProperty(propertyData);
    } catch (Exception e){
        System.out.println("System Exception in addProperty");
        return false;
    }
}

/**
 * This method is used by the client to add department data
 * to the database
 *
 * @param      departmentData The struct that holds department
 *              information
 * @return     boolean true or false
 * @exception  none
 */
public boolean addDepartment(Navy.departmentStruct departmentData ){
    try {
        return myDBHelper.addDepartment(departmentData);
    } catch (Exception e){
        System.out.println("System Exception in addDepartment");
        return false;
    }
}

/**
 * This method is used by the client to add division data
 * to the database
 *
 * @param      divisionData The struct that holds division
 *              information
 * @return     boolean true or false
 * @exception  none
 */
public boolean addDivision(Navy.divisionStruct divisionData ){

```

```

    try {
        return myDBHelper.addDivision(divisionData);
    } catch (Exception e){
        System.out.println("System Exception in addDivision");
        return false;
    }
}

/**
 * This method is used by the client to add maintenance data
 * to the database
 *
 * @param      maintenanceData The struct that holds maintenance
 *              information
 * @return      boolean true or false
 * @exception   none
 */
public boolean addMaintLog(Navy.maintenanceStruct maintenanceData ){
    try {
        return myDBHelper.addMaintLog(maintenanceData);
    } catch (Exception e){
        System.out.println("System Exception in addMaintLog");
        return false;
    }
}

/**
 * This method is used by the client to add sailor data
 * to the database
 *
 * @param      sailorData The struct that holds sailor information
 * @return      boolean true or false
 * @exception   none
 */
public boolean addSailor(Navy.sailorStruct sailorData ){
    try {
        return myDBHelper.addSailor(sailorData);
    } catch (Exception e){
        System.out.println("System Exception in addSailor");
        return false;
    }
}
}

```



```

/**
 * This method is used by the client to add sea mission data
 * to the database
 *
 * @param      seamissionData The struct that holds sea mission
 *              information
 * @return      boolean true or false
 * @exception   none
 */
public boolean addSeaMission(Navy.seamissionStruct seamissionData ){
    try {
        return myDBHelper.addSeaMission(seamissionData);
    } catch (Exception e){
        System.out.println("System Exception in addSeaMission");
        return false;
    }
}

```

```

/**
 * This method is used by the client to add air mission data
 * to the database
 *
 * @param      airmissionData The struct that holds air mission
 *              information
 * @return      boolean true or false
 * @exception   none
 */
public boolean addAirMission(Navy.airmissionStruct airmissionData){
    try {
        return myDBHelper.addAirMission(airmissionData);
    } catch (Exception e){
        System.out.println("System Exception in addAirMission");
        return false;
    }
}

```

```

/**
 * This method is used by the client to add department data
 * to the database
 *
 * @param      departmentData The struct that holds department
 *              information
 * @return      boolean true or false
 * @exception   none
 */
public boolean changeDeptInfo(Navy.departmentStruct departmentData)
{
    try {
        return myDBHelper.changeDeptInfo(departmentData);
    } catch (Exception e) {

```

```

        System.out.println("System Exception in changeDeptInfo");
        return false;
    }
}

/**
 * This method is used by the client to add division data
 * to the database
 *
 * @param      divisionData The struct that holds division
 *              information
 * @return      boolean true or false
 * @exception   none
 */
public boolean changeDivisionInfo(Navy.divisionStruct divisionData)
{
    try {
        return myDBHelper.addDivision(divisionData);
    } catch (Exception e){
        System.out.println("System Exception in changeDivisionInfo");
        return false;
    }
}

/**
 * This method is used by the client to log onto the database
 *
 * @param      String userName
 * @param      String password
 * @return      boolean true or false
 * @exception   none
 */
public boolean getAuthorization(java.lang.String userName,
                               java.lang.String password){
    boolean result = false;

    try{
        result = myDBHelper.getAuthorization(userName, password);
        return result;
    } catch (Exception ex){
        System.out.println("System Exception in getAuthorization");
        return false;
    }
}

```

```

/**
 * This method is used by the client to get information about the
 * family members of a sailor
 *
 * @param sailorSsn Social security number of the sailor
 * @param lastName Last Name of the sailor
 * @return struct that contains information about the family
 *         members of the sailor
 * @exception none
 */
public Navy.familydescStruct[] getFamilyMembers(java.lang.String
                                                sailorSsn,
                                                java.lang.String
                                                lastName)
{
    try {
        Navy.familydescStruct[] familyData;
        Vector families =
            myDBHelper.getFamilyMembers(sailorSsn,lastName);
        familyData = new Navy.familydescStruct[families.size()];
        families.copyInto(familyData);
        return(familyData);
    } catch (Exception e){
        System.out.println("System Exception in getFamilyMembers");
        return null;
    }
}

/**
 * This method is used by the client to get information about the
 * spouse of a sailor
 *
 * @param sailorSsn Social security number of the sailor
 * @param lastName Last Name of the sailor
 * @return struct that contains information about the
 *         spouse of the sailor
 * @exception none
 */
public Navy.spousedescStruct getSpouse( java.lang.String sailorSsn,
                                         java.lang.String lastName)
{
    try {
        Navy.spousedescStruct[] spouseData;
        Vector spouse = myDBHelper.getSpouse(sailorSsn,lastName);
        spouseData = new Navy.spousedescStruct[spouse.size()];
        spouse.copyInto(spouseData);
        return(spouseData[0]);
    } catch (Exception e){
        System.out.println("System Exception in getSpouse");
        return null;
    }
}

```

```
}
```

```
/**
 * This method is used by the client to get information about the
 * leave periods of a sailor
 *
 * @param      sailorSsn Social security number of the sailor
 * @param      lastName  Last Name of the sailor
 * @return     struct that contains information about the
 *             leave periods of the sailor
 * @exception  none
 */
public Navy.leavedescStruct[] getLeaveData(java.lang.String
                                           sailorSsn,
                                           java.lang.String lastName)
{
    try {
        Navy.leavedescStruct[] leaveData;
        Vector leaves = myDBHelper.getLeaveData(sailorSsn,lastName);
        leaveData = new Navy.leavedescStruct[leaves.size()];
        leaves.copyInto(leaveData);
        return(leaveData);
    } catch (Exception e) {
        System.out.println("System Exception in getLeaveData");
        return null;
    }
}
```

```
/**
 * This method is used by the client to get information about the
 * prt results of a sailor from the database
 *
 * @param      sailorSsn Social security number of the sailor
 * @param      lastName  Last Name of the sailor
 * @return     struct that contains information about the
 *             prt results of the sailor
 * @exception  none
 */
public Navy.prtdescStruct[] getPrtResults(java.lang.String
                                           sailorSsn,
                                           java.lang.String lastName)
{
    try {
        Navy.prtdescStruct[] prtData;
        Vector prtresults =
            myDBHelper.getPrtResults(sailorSsn,lastName);
        prtData = new Navy.prtdescStruct[prtresults.size()];
        prtresults.copyInto(prtData);
        return(prtData);
    } catch (Exception e){
```

```

        System.out.println("System Exception in getPrtResults");
        return null;
    }
}

/**
 * This method is used by the client to get information about the
 * deployment days of a sailor
 *
 * @param sailorSsn Social security number of the sailor
 * @param lastName Last Name of the sailor
 * @param startDate The beginning of the search period
 * @param stopDate The end of the search period
 * @return struct that contains information about the
 *         deployment days of the sailor
 * @exception none
 */
public Navy.deploymentStruct[] getDaysDeployed(java.lang.String
                                              sailorSsn,
                                              java.lang.String
                                              lastName,
                                              java.lang.String
                                              startDate,
                                              java.lang.String
                                              stopDate)
{
    try {
        Navy.deploymentStruct[] deploymentData;
        Vector deployments =
            myDBHelper.getDaysDeployed(sailorSsn, lastName,
                                      startDate, stopDate);
        System.out.println("Returning vector"+deployments);
        deploymentData =
            new Navy.deploymentStruct[deployments.size()];
        deployments.copyInto(deploymentData);
        return(deploymentData);
    } catch (Exception e){
        System.out.println("System Exception in getDaysDeployed");
        return null;
    }
}

```

```

/**
 * This method is used by the client to get information about the
 * properties of a command
 *
 * @param      commandId Specific command
 * @return      struct that contains information about the
 *              deployment days of the sailor
 * @exception   none
 */
public Navy.propertydescStruct[] getPropertyList(java.lang.String
                                                commandId)
{
    try {
        Navy.propertydescStruct[] propertyData;
        Vector properties = myDBHelper.getPropertyList(commandId);
        propertyData = new Navy.propertydescStruct[properties.size()];
        properties.copyInto(propertyData);
        return(propertyData);
    } catch (Exception e) {
        System.out.println("System Exception in getPropertyList");
        return null;
    }
}

```

```

/**
 * This method is used by the client to get information about the
 * deployment days of a specific sailor
 *
 * @param      commandId Specific command
 * @param      sex gender of sailors
 * @return      struct that contains information about the
 *              deployment days of the sailor
 * @exception   none
 */
public Navy.demographicStruct[] getComDemByGender(java.lang.String
                                                commandId,
                                                java.lang.String
                                                sex)
{
    try {
        Navy.demographicStruct[] demographicData;
        Vector demographics =
            myDBHelper.getComDemByGender(commandId,sex);
        demographicData =
            new Navy.demographicStruct[demographics.size()];
        demographics.copyInto(demographicData);
        return(demographicData);
    } catch (Exception e) {
        System.out.println("System Exception in getComDemByGender");
        return null;
    }
}

```

```
}
```

```
/**
```

```
 * This method is used by the client to get information about the  
 * demographic data of a command
```

```
 *
```

```
 * @param      commandId ID Number for the command
```

```
 * @param      nec1      Designation for NEC1
```

```
 * @param      nec2      Designation for NEC2
```

```
 * @param      nec3      Designation for NEC3
```

```
 * @return     struct that contains information about the
```

```
 *             demographics of the command
```

```
 * @exception  none
```

```
 */
```

```
public Navy.demographicStruct[] getComDemByNec(java.lang.String  
                                              commandId,  
                                              java.lang.String  
                                              nec1,  
                                              java.lang.String  
                                              nec2,  
                                              java.lang.String  
                                              nec3)
```

```
{
```

```
    try {
```

```
        Navy.demographicStruct[] demographicData;
```

```
        Vector demographics = myDBHelper.getComDemByNec
```

```
            (commandId,nec1,nec2,nec3);
```

```
        demographicData =
```

```
            new Navy.demographicStruct[demographics.size()];
```

```
        demographics.copyInto(demographicData);
```

```
        return(demographicData);
```

```
    } catch (Exception e) {
```

```
        System.out.println("System Exception in getComDemByGender");
```

```
        return null;
```

```
    }
```

```
}
```

```
/**
```

```
 * This method is used by the client to get information about the
```

```
 * demographic data of a command according to paygrade
```

```
 *
```

```
 * @param      commandId ID Number for the command
```

```
 * @param      paygrade  The value that defines the range of rank
```

```
 * @return     struct that contains information about the
```

```
 *             demographics of the command
```

```
 * @exception  none
```

```
 */
```

```
public Navy.demographicStruct[] getComDemByRank(java.lang.String  
                                              commandId,  
                                              java.lang.String
```

```

                                paygrade)
{
    try {
        Navy.demographicStruct[] demographicData;
        Vector demographics =
            myDBHelper.getComDemByRank(commandId,paygrade);
        demographicData =
            new Navy.demographicStruct[demographics.size()];
        demographics.copyInto(demographicData);
        return(demographicData);
    } catch (Exception e) {
        System.out.println("System Exception in getComDemByGender");
        return null;
    }
}

/**
 * This method is used by the client to get information about the
 * emergency list of a command
 *
 * @param      commandId ID Number for the command
 * @return     struct that contains information about the
 *             emergency list of the command
 * @exception  none
 */
public Navy.emergencyStruct[] getEmergencyList(java.lang.String
                                commandId)
{
    try {
        Navy.emergencyStruct[] emergencyData;
        Vector emergencyList = myDBHelper.getEmergencyList(commandId);
        emergencyData =
            new Navy.emergencyStruct[emergencyList.size()];
        emergencyList.copyInto(emergencyData);
        return(emergencyData);
    } catch (Exception e){
        System.out.println("System Exception in getEmergencyList");
        return null;
    }
}

```



```

/**
 * This method is used by the client to get information about the
 * sea missions conducted by a specific platform and of a specific
 * type in a given period of time
 *
 * @param      platform      The platform type
 * @param      typeMission   The type of the mission
 * @param      missionArea   Area of interest
 * @param      startDate     Beginning date of the period
 * @param      stopDate      Ending date of the period
 * @return      struct that contains information about the
 *              sea missions flown by the platform
 * @exception   none
 */
public Navy.seamissiondescStruct[] getSeaMissions(
                                java.lang.String
                                platform,
                                java.lang.String
                                typeMission,
                                java.lang.String
                                missionArea,
                                java.lang.String
                                startDate,
                                java.lang.String
                                stopDate)
{
    try {
        Navy.seamissiondescStruct[] seamisdescData;
        Vector missions = myDBHelper.getSeaMissions(platform,
                                                    typeMission,
                                                    missionArea,
                                                    startDate,
                                                    stopDate);

        seamisdescData =
            new Navy.seamissiondescStruct[missions.size()];
        missions.copyInto(seamisdescData);
        return(seamisdescData);
    } catch (Exception e){
        System.out.println("System Exception in getSeaMissions");
        return null;
    }
}

```

```

/**
 * This method is used by the client to get information about the
 * sea missions conducted by a specific platform
 * in a given period of time
 *
 * @param      platform      The platform type
 * @param      startDate     Beginning date of the period
 * @param      stopDate      Ending date of the period
 * @return      struct that contains information about the
 *              sea missions flown by the platform
 * @exception   none
 */
public Navy.seamissiondescStruct[] getSeaMissionsByPlatform(
                                                    java.lang.String
                                                    platform,
                                                    java.lang.String
                                                    startDate,
                                                    java.lang.String
                                                    stopDate)
{
    try {
        Navy.seamissiondescStruct[] seamisdescData;
        Vector missions = myDBHelper.getSeaMissionsByPlatform
                                (platform,startDate, stopDate);

        seamisdescData =
            new Navy.seamissiondescStruct[missions.size()];
        missions.copyInto(seamisdescData);
        return(seamisdescData);
    } catch (Exception e){
        System.out.println("System Exception in
                                GetSeaMissionsByPlatform");
        return null;
    }
}

/**
 * This method is used by the client to get information about the
 * sea missions conducted in a certain mission area
 * in a given period of time
 *
 * @param      missionArea   Area of interest
 * @param      startDate     Beginning date of the period
 * @param      stopDate      Ending date of the period
 * @return      struct that contains information about the
 *              sea missions flown that meets criteria
 * @exception   none
 */
public Navy.seamissiondescStruct[] getSeaMissionsByArea(
                                                    java.lang.String
                                                    missionArea,

```

```

                                java.lang.String
                                startDate,
                                java.lang.String
                                stopDate){
try {
    Navy.seamissiondescStruct[] seamisdescData;
    Vector missions = myDBHelper.getSeaMissionsByArea
        (missionArea, startDate, stopDate);

    seamisdescData =
        new Navy.seamissiondescStruct[missions.size()];
    missions.copyInto(seamisdescData);
    return(seamisdescData);
} catch (Exception e) {
    System.out.println("System Exception in
                        getSeaMissionsByArea");
    return null;
}
}

```

```

/**
 * This method is used by the client to get information about the
 * sea missions conducted of a specified type
 * in a given period of time
 *
 * @param      typeMission  The type of the mission
 * @param      startDate    Beginning date of the period
 * @param      stopDate     Ending date of the period
 * @return     struct that contains information about the
 *             sea missions flown that meets criteria
 * @exception  none
 */

```

```

public Navy.seamissiondescStruct[] getSeaMissionsByType(
                                java.lang.String
                                typeMission,
                                java.lang.String
                                startDate,
                                java.lang.String
                                stopDate)
{
    try {
        Navy.seamissiondescStruct[] seamisdescData;
        Vector missions = myDBHelper.getSeaMissionsByType
            (typeMission, startDate, stopDate);

        seamisdescData =
            new Navy.seamissiondescStruct[missions.size()];
        missions.copyInto(seamisdescData);
        return(seamisdescData);
    } catch (Exception e) {
        System.out.println("System Exception in

```

```

        getSeaMissionsByArea");
    return null;
}

/**
 * This method is used by the client to get information about the
 * sea missions conducted in the exercise specified
 * in a given period of time
 *
 * @param      exerciseName Name of the exercise
 * @param      startDate    Beginning date of the period
 * @param      stopDate     Ending date of the period
 * @return     struct that contains information about the
 *             sea missions flown that meets criteria
 * @exception  none
 */
public Navy.seamissiondescStruct[] getSeaMissionsByExercise(
    java.lang.String
    exerciseName,
    java.lang.String
    startDate,
    java.lang.String
    stopDate)
{
    try {
        Navy.seamissiondescStruct[] seamisdescData;
        Vector missions = myDBHelper.getSeaMissionsByExercise
            (exerciseName, startDate,
            stopDate);

        seamisdescData =
            new Navy.seamissiondescStruct[missions.size()];
        missions.copyInto(seamisdescData);
        return(seamisdescData);
    } catch (Exception e) {
        System.out.println("System Exception in
            getSeaMissionsByExercise");
        return null;
    }
}

```

```

/**
 * This method is used by the client to get information about the
 * sea missions conducted by a specific platform
 * in a given period of time
 *
 * @param      platform      The platform type
 * @param      startDate     Beginning date of the period
 * @param      stopDate      Ending date of the period
 * @return      struct that contains information about the
 *              sea missions flown that meets criteria
 * @exception   none
 */
public Navy.seamissiondescStruct[] getSeaMissionsByPlatformName
                                   (java.lang.String
                                   platform,
                                   java.lang.String
                                   startDate,
                                   java.lang.String
                                   stopDate)
{
    try {
        Navy.seamissiondescStruct[] seamisdescData;
        Vector missions = myDBHelper.getSeaMisByPlatformName
                                   (platform, startDate, stopDate);

        seamisdescData =
            new Navy.seamissiondescStruct[missions.size()];
        missions.copyInto(seamisdescData);
        return(seamisdescData);
    } catch (Exception e){
        System.out.println("System Exception in
                           getSeaMissionsByPlatformName");
        return null;
    }
}

/**
 * This method is used by the client to get information about the
 * air missions flown by a specific squadron and of a specific
 * mission type in a given period of time
 *
 * @param      squadron      The squadron name
 * @param      platform      The platform type
 * @param      typeMission    The type of the mission
 * @param      missionArea    Area of interest
 * @param      startDate     Beginning date of the period
 * @param      stopDate      Ending date of the period
 * @return      struct that contains information about the
 *              air missions flown that meets criteria
 * @exception   none
 */

```

```

public Navy.airmissiondescStruct[] getAirMissions(
    java.lang.String
    squadron,
    java.lang.String
    platform,
    java.lang.String
    typeMission,
    java.lang.String
    missionArea,
    java.lang.String
    startDate,
    java.lang.String
    stopDate)
{
    try {
        Navy.airmissiondescStruct[] airmisdescData;
        Vector missions = myDBHelper.getAirMissions
            ( squadron, platform, typeMission, missionArea, startDate,
              stopDate);

        airmisdescData =
            new Navy.airmissiondescStruct[missions.size()];
        missions.copyInto(airmisdescData);
        return(airmisdescData);
    } catch (Exception e) {
        System.out.println("System Exception in getAirMissions");
        return null;
    }
}

```

```

/**
 * This method is used by the client to get information about the
 * air missions flown by a specific platform
 * in a given period of time
 *
 * @param      platform      The platform type
 * @param      startDate     Beginning date of the period
 * @param      stopDate      Ending date of the period
 * @return     struct that contains information about the
 *             air missions flown that meets criteria
 * @exception  none
 */

```

```

public Navy.airmissiondescStruct[] getAirMisByPlatformNo(
    java.lang.String
    platform,
    java.lang.String
    startDate,
    java.lang.String
    stopDate)
{
    try {

```

```

        Navy.airmissiondescStruct[] airmisdescData;
        Vector missions = myDBHelper.getAirMisByPlatformNo
        ( platform, startDate, stopDate);

        airmisdescData =
            new Navy.airmissiondescStruct[missions.size()];
        missions.copyInto(airmisdescData);
        return(airmisdescData);
    } catch (Exception e) {
        System.out.println("System Exception in
                           getAirMisByPlatformNo");
        return null;
    }
}

/**
 * This method is used by the client to get information about the
 * air missions flown in a specified area
 * in a given period of time
 *
 * @param      missionArea  Area of interest
 * @param      startDate    Beginning date of the period
 * @param      stopDate     Ending date of the period
 * @return     struct that contains information about the
 *             air missions flown that meets criteria
 * @exception  none
 */
public Navy.airmissiondescStruct[] getAirMisByArea(
                                                    java.lang.String
                                                    missionArea,
                                                    java.lang.String
                                                    startDate,
                                                    java.lang.String
                                                    stopDate)
{
    try {
        Navy.airmissiondescStruct[] airmisdescData;
        Vector missions = myDBHelper.getAirMisByArea
        ( missionArea, startDate, stopDate);

        airmisdescData =
            new Navy.airmissiondescStruct[missions.size()];
        missions.copyInto(airmisdescData);
        return(airmisdescData);
    } catch (Exception e) {
        System.out.println("System Exception in getAirByArea");
        return null;
    }
}

```

```

/**
 * This method is used by the client to get information about the
 * air missions flown of a specified mission type
 * in a given period of time
 *
 * @param      typeOfMission  The type of the mission
 * @param      startDate      Beginning date of the period
 * @param      stopDate       Ending date of the period
 * @return     struct that contains information about the
 *             air missions flown that meets criteria
 * @exception  none
 */
public Navy.airmissiondescStruct[] getAirMisByType(
                                                    java.lang.String
                                                    typeOfMission,
                                                    java.lang.String
                                                    startDate,
                                                    java.lang.String
                                                    stopDate)
{
    try {
        Navy.airmissiondescStruct[] airmisdescData;
        Vector missions = myDBHelper.getAirMisByType
            ( typeOfMission, startDate, stopDate);

        airmisdescData =
            new Navy.airmissiondescStruct[missions.size()];
        missions.copyInto(airmisdescData);
        return(airmisdescData);
    } catch (Exception e) {
        System.out.println("System Exception in getAirMisByType");
        return null;
    }
}

/**
 * This method is used by the client to get information about the
 * air missions flown in a specified exercise
 * in a given period of time
 *
 * @param      exerciseName  Name of the exercise
 * @param      startDate      Beginning date of the period
 * @param      stopDate       Ending date of the period
 * @return     struct that contains information about the
 *             air missions flown that meets criteria
 * @exception  none
 */
public Navy.airmissiondescStruct[] getAirMisByExercise(
                                                    java.lang.String
                                                    exerciseName,
                                                    java.lang.String

```



```

                                startDate,
                                java.lang.String
                                stopDate)
{
    try {
        Navy.airmissiondescStruct[] airmisdescData;
        Vector missions = myDBHelper.getAirMisByExercise
            ( exerciseName, startDate, stopDate);

        airmisdescData =
            new Navy.airmissiondescStruct[missions.size()];
        missions.copyInto(airmisdescData);
        return(airmisdescData);
    } catch (Exception e) {
        System.out.println("System Exception in getAirMisByExercise");
        return null;
    }
}

```

```

/**
 * This method is used by the client to get information about the
 * air missions flown by a specific squadron
 * in a given period of time
 *
 * @param      squadronName      The squadron name
 * @param      startDate          Beginning date of the period
 * @param      stopDate           Ending date of the period
 * @return      struct that contains information about the
 *              air missions flown that meets criteria
 * @exception   none
 */
public Navy.airmissiondescStruct[] getAirMisBySquadron(
                                java.lang.String
                                squadronName,
                                java.lang.String
                                startDate,
                                java.lang.String
                                stopDate)
{
    try {
        Navy.airmissiondescStruct[] airmisdescData;
        Vector missions = myDBHelper.getAirMisBySquadron
            ( squadronName , startDate, stopDate);

        airmisdescData =
            new Navy.airmissiondescStruct[missions.size()];
        missions.copyInto(airmisdescData);
        return(airmisdescData);
    } catch (Exception e){
        System.out.println("System Exception in getAirMisBySquadron");
        return null;
    }
}

```

```

    }
}

/**
 * This method is used by the client to get information about the
 * maintenance data stored for the inventory of a command
 * in a given period of time
 *
 * @param      commandId      ID Number of the command
 * @param      departmentId  ID Number of the department
 * @param      startDate      Beginning date of the period
 * @param      stopDate       Ending date of the period
 * @return      struct that contains information about the
 *              maintenance data that meets criteria
 * @exception   none
 */
public Navy.commaintenanceStruct[] getMaintByCommand(
    java.lang.String
        commandId,
    java.lang.String
        departmentId,
    java.lang.String
        startDate,
    java.lang.String
        stopDate){
    try {
        Navy.commaintenanceStruct[] commaintenanceData;
        Vector maintenanceList = myDBHelper.getMaintByCommand
            (commandId, departmentId, startDate, stopDate);

        commaintenanceData = new Navy.commaintenanceStruct
            [maintenanceList.size()];
        maintenanceList.copyInto(commaintenanceData);
        return(commaintenanceData);
    } catch (Exception e){
        System.out.println("System Exception in getMaintByCommand");
        return null;
    }
}

/**
 * This method is used by the client to get information about the
 * maintenance data stored about an item in the database
 *
 * @param      itemId      ID Number of the item
 * @return      struct that contains information about the
 *              maintenance data that meets criteria
 * @exception   none
 */
public Navy.itemmaintenanceStruct[] getMaintByItem(

```

```

                                java.lang.String
                                itemId)
{
    try {
        Navy.itemmaintenanceStruct[] itemmaintenanceData;
        Vector maintenanceList = myDBHelper.getMaintByItem(itemId);

        itemmaintenanceData = new Navy.itemmaintenanceStruct
                                [maintenanceList.size()];
        maintenanceList.copyInto(itemmaintenanceData);
        return(itemmaintenanceData);
    } catch (Exception e) {
        System.out.println("System Exception in getMaintByItem");
        return null;
    }
}

} // End NavSecImpl.java

```

```

//-----
// Filename      : DBHelper.java
// Author        : Murat Akbay & Steve Lewis
// Date         : 10/09/1998
// Compiler      : JDK1.1.6 with Symantec JIT Compiler
//-----

/**
 * Server component that uses JDBC to access the back-end database and
 * execute the queries on that database, the results of the query are
 * returned as a vector of structs to the calling class
 *
 * @author Murat Akbay & Steve Lewis
 */

import java.net.URL;
import java.sql.*;
import java.util.*;

public class DBHelper{

    Connection con;
    Driver driver = null;
    ResultSet rs;
    PreparedStatement pstmt;
    boolean debug = false;

    /**
     * This method is used to open a connection to the database through
     * port 1433
     *
     * @param      address of computer where database is held, database
     *             name, user ID, and password
     * @return     none
     * @exception  system exception to catch all problems
     */
    public void connect(String address, String db, String id, String pw)
        throws Exception
    {
        try
        {
            //Load the fastforward jdbc driver
            Class.forName("connect.microsoft.MicrosoftDriver");
            String url= "jdbc:ff-microsoft://" + address + ":1433/" + db;
            System.out.println("Connecting to" + url );
            con = DriverManager.getConnection( url , id , pw);
        }
        catch(Exception e)
        {
            System.err.println("System Exception in connect");
            System.err.println(e);
            throw e;
        }
    }
}

```

```

    }
}

```

```

/**
 * This method is used to close the connection with the database
 *
 * @param      none
 *
 * @return      none
 * @exception  system exception to catch all problems
 */

```

```

public void closeConnection() throws Exception

```

```

{
    try
    {
        System.err.println("Closing connection");
        con.close();
    }
    catch(Exception e)
    {
        System.err.println("System Exception in closeConnection");
        System.err.println(e);
        throw e;
    }
}

```

```

/**
 * This method is used to query the database for a specific sailors
 * address information based on the input of an SSN.
 *
 * @param      adressData  address struct that contains all applicable
 *                          information
 *
 * @return      boolean
 * @exception  system exception to catch all problems
 */

```

```

public boolean addAdress( Navy.adressStruct adressData)
                        throws Exception

```

```

{
    try
    {
        pstmt = con.prepareStatement(
            "UPDATE SAILOR SET "+
            "ADDRESS = ? , "+
            "CITY = ? , "+
            "STATEORPROVINCE = ? , "+
            "COUNTRY = ? , "+
            "POSTALCODE = ? , "+

```

```

        "HOMEPHONE = ? "+
        "WHERE Social_Security_Number = ? ");
pstmt.setString( 1, adressData.adress);
pstmt.setString( 2, adressData.city);
pstmt.setString( 3, adressData.state);
pstmt.setString( 4, adressData.country);
pstmt.setString( 5, adressData.postalCode);
pstmt.setString( 6, adressData.homePhone);
pstmt.setString( 7, adressData.ssn);

pstmt.executeUpdate();
pstmt.close();
con.commit();
if(debug == true){
    System.out.println("Query run successfully");
}
return true;
}
catch(Exception e)
{
    System.err.println("System Exception in addAdress");
    System.err.println(e);
    throw e;
}
}

/**
 * This method is used to add a family member to the database
 * regarding a specific sailor
 *
 * @param    family struct that contains all applicable information
 *
 * @return    boolean
 * @exception system exception to catch all problems
 */
public boolean addFamily( Navy.familyStruct familyData)
                        throws Exception
{

    java.sql.Date bdate= java.sql.Date.valueOf(familyData.sdate);
    try
    {
        pstmt = con.prepareStatement(
            " INSERT INTO FAMILY_MEMBERS "+
            "( FAMILY_MEMBER_SOCIAL_SECURITY_, "+
            " NAME, BIRTHDATE , SEX, "+
            " SAILOR_SOCIAL_SECURITY_NUMBER) "+
            " VALUES ( ? ,? ,? ,? ,? ) ");

        pstmt.setString( 1, familyData.ssn);
        pstmt.setString( 2, familyData.name);
    }
}

```

```

        pstmt.setDate( 3, bdate);
        pstmt.setString( 4, familyData.sex);
        pstmt.setString( 5, familyData.sailorSsn);
        pstmt.executeUpdate();
        pstmt.close();
        con.commit();
        if(debug == true){
            System.out.println("Query run successfully");
        }
        return true;
    }
    catch(Exception e)
    {
        System.err.println("System Exception in addFamily");
        System.err.println(e);
        throw e;
    }
}

```

```

/**
 * This method is used to add a spouse of a specific sailor to the
 * database
 *
 * @param      spouse struct that contains all applicable information
 *
 * @return      boolean
 * @exception  system exception to catch all problems
 */
public boolean addSpouses( Navy.spouseStruct spouseData)
                        throws Exception
{
    java.sql.Date birthDate= java.sql.Date.valueOf(spouseData.bdate);
    java.sql.Date uDate= java.sql.Date.valueOf(spouseData.update);
    try
    {
        pstmt = con.prepareStatement(
            " INSERT INTO SPOUSES "+
            "( SPOUSE_SOCIAL_SECURITY_NUMBER, "+
            " SPOUSE_NAME, SPOUSE_DATE_OF_BIRTH , SPOUSE_GENDER, "+
            " SAILOR_SOCIAL_SECURITY_NUMBER , "+
            " DATEUPDATED) "+
            " VALUES ( ? , ? , ? , ? , ? , ? ) ";
        pstmt.setString( 1, spouseData.ssn);
        pstmt.setString( 2, spouseData.name);
        pstmt.setDate( 3, birthDate);
        pstmt.setString( 4, spouseData.gender);
        pstmt.setString( 5, spouseData.sailorSsn);
        pstmt.setDate( 6, uDate);
        pstmt.executeUpdate();
        pstmt.close();
    }
}

```

```

        con.commit();
        if(debug == true){
            System.out.println("Query run successfully");
        }
        return true;
    }
    catch(Exception e)
    {
        System.err.println("System Exception in addSpouses");
        System.err.println(e);
        throw e;
    }
}

/**
 * This method is used to add the PRT data of a specific sailor to
 * the database
 *
 * @param      PRT struct that contains all applicable information
 *
 * @return     boolean
 * @exception  system exception to catch all problems
 */
public boolean addPrt( Navy.prtStruct prtData )throws Exception
{
    java.sql.Date prtDate= java.sql.Date.valueOf(prtData.pdate);
    try
    {
        pstmt = con.prepareStatement(
            " INSERT INTO PRT_DATA( COMMAND_ID, "+
            " PRT_IDENTIFIER, HEIGHT, WEIGHT , "+
            " PUSHUPS, SITUPS , RUN_TIME, PRT_DATE, "+
            " STATUS, SOCIAL_SECURITY_NUMBER) "+
            " VALUES ( ? , ? , ? , ? , ? , ? , ? , ? , ? , ? ) ");

        pstmt.setString( 1, prtData.commandId);
        pstmt.setString( 2, prtData.identifier);
        pstmt.setString( 3, prtData.height);
        pstmt.setString( 4, prtData.weight);
        pstmt.setString( 5, prtData.pushUps);
        pstmt.setString( 6, prtData.sitUps);
        pstmt.setString( 7, prtData.runTime);
        pstmt.setDate( 8, prtDate);
        pstmt.setString( 9, prtData.status);
        pstmt.setString(10, prtData.ssn);
        pstmt.executeUpdate();
        pstmt.close();
        con.commit();
        if(debug == true){
            System.out.println("Query run successfully");
        }
    }
}

```



```

        return true;
    }
    catch(Exception e)
    {
        System.err.println("System Exception in addPrt");
        System.err.println(e);
        throw e;
    }
}

/**
 * This method is used to modify a members leave data in
 * the database
 *
 * @param      leave struct that contains all applicable information
 *
 * @return      boolean
 * @exception  system exception to catch all problems
 */
public boolean addLeave( Navy.leaveStruct leaveData)
                        throws Exception
{
    java.sql.Date dateDepart =
        java.sql.Date.valueOf(leaveData.departDate);
    java.sql.Date dateReturn =
        java.sql.Date.valueOf(leaveData.returnDate);

    try
    {
        pstmt = con.prepareStatement(
            " INSERT INTO LEAVE( COMMAND_ID,"+
            " LEAVE_CONTROL_NUMBER, "+
            " TYPE_OF_LEAVE, DATE_OF_DEPARTURE,"+
            " DATE_OF_RETURN ,"+
            " SOCIAL_SECURITY_NUMBER) "+
            " VALUES ( ? , ? , ? , ? , ? , ? ) ");

        pstmt.setString( 1, leaveData.commandId);
        pstmt.setString( 2, leaveData.leaveControlNumber);
        pstmt.setString( 3, leaveData.typeOfLeave);
        pstmt.setDate( 4, dateDepart);
        pstmt.setDate( 5, dateReturn);
        pstmt.setString( 6, leaveData.ssn);
        pstmt.executeUpdate();
        pstmt.close();
        con.commit();
        if(debug == true){
            System.out.println("Query run successfully");
        }
        return true;
    }
}

```

```

    }
    catch(Exception e)
    {
        System.err.println("System Exception in addLeave");
        System.err.println(e);
        throw e;
    }
}

/**
 * This method is used to add a new NEC to a specific sailor in
 * the database
 *
 * @param      NEC struct that contains all applicable information
 *
 * @return     boolean
 * @exception  system exception to catch all problems
 */
public boolean addNec( Navy.necStruct necData) throws Exception
{
    java.sql.Date lastUpDate =
        java.sql.Date.valueOf(necData.lastDate);

    try
    {
        pstmt = con.prepareStatement(
            " UPDATE SAILOR SET PRIMARY_NEC_DESIGNATOR = ? , "+
            " SECONDARY_NEC = ? , TERTIARY_NEC = ? , "+
            " LAST_UPDATED = ? "+
            " WHERE SOCIAL_SECURITY_NUMBER = ? ");

        pstmt.setString( 1, necData.primaryNec);
        pstmt.setString( 2, necData.secondaryNec);
        pstmt.setString( 3, necData.tertiaryNec);
        pstmt.setDate( 4, lastUpDate);
        pstmt.setString( 5, necData.ssn);
        pstmt.executeUpdate();
        pstmt.close();
        con.commit();
        if(debug == true){
            System.out.println("Query run successfully");
        }
        return true;
    }
    catch(Exception e)
    {
        System.err.println("System Exception in addNec");
        System.err.println(e);
        throw e;
    }
}

```

```

/**
 * This method is used to add new minor property data to
 * the database
 *
 * @param      property struct that contains all applicable
 *              information
 *
 * @return      boolean
 * @exception   system exception to catch all problems
 */
public boolean addProperty( Navy.propertyStruct propertyData)
                                throws Exception
{
    int propertyPrice =
        (Integer.valueOf(propertyData.price)).intValue();
    java.util.Date dateNow = new java.util.Date();
    java.sql.Date sqlDate = new java.sql.Date(dateNow.getYear(),
        dateNow.getMonth(), dateNow.getDay());
    java.sql.Date datePurchase =
        java.sql.Date.valueOf(propertyData.purchaseDate);

    try
    {
        pstmt = con.prepareStatement(
            " INSERT INTO PROPERTIES_LISTING( PROPERTY_ID, "+
            " PROPERTY_DESCRIPTION, "+
            " DATE_PURCHASED, PRICE, COMMAND_ID, SUBCUSTODIED, "+
            " PURCHASER, INVENTORY, "+
            " LAST_INVENTORY_DATE, PURCHASE_CATEGORY )" +
            " VALUES ( ? , ? , ? , ? , ? , ? , ? , ? , ? , ? ) ";

        pstmt.setString( 1, propertyData.propertyId);
        pstmt.setString( 2, propertyData.propertyDesc);
        pstmt.setDate( 3, datePurchase);
        pstmt.setInt( 4, propertyPrice);
        pstmt.setString( 5, propertyData.commandId);
        pstmt.setString( 6, propertyData.purchaser);
        pstmt.setString( 7, propertyData.purchaser);
        pstmt.setString( 8, propertyData.purchaser);
        pstmt.setDate( 9, sqlDate);
        pstmt.setString( 10, propertyData.purchaseCategory);
        pstmt.executeUpdate();
        pstmt.close();
        con.commit();
        if(debug == true){
            System.out.println("Query run successfully");
        }
        return true;
    }
    catch(Exception e)
    {
        System.err.println("System Exception in addProperty");
        System.err.println(e);
    }
}

```

```

        throw e;
    }
}

/**
 * This method is used to add a new department to
 * the database
 *
 * @param    department struct that contains all applicable
 *           information
 *
 * @return    boolean
 * @exception system exception to catch all problems
 */
public boolean addDepartment( Navy.departmentStruct departmentData)
                                throws Exception
{
    try
    {
        pstmt = con.prepareStatement(
            " INSERT INTO DEPARTMENT( "+
            " COMMAND_ID, DEPARTMENT_ID, TITLE ,"+
            " DEPARTMENT_HEAD_SSN, DEPARTMENT_CHIEF_SSN, "+
            " TELEPHONE, EMAIL )" +
            " VALUES ( ? ,? ,? ,? ,? ,? ,? ) ");

        pstmt.setString( 1, departmentData.commandId);
        pstmt.setString( 2, departmentData.departmentId);
        pstmt.setString( 3, departmentData.title);
        pstmt.setString( 4, departmentData.headSsn);
        pstmt.setString( 5, departmentData.chiefSsn);
        pstmt.setString( 6, departmentData.telephone);
        pstmt.setString( 7, departmentData.email);
        pstmt.executeUpdate();
        pstmt.close();
        con.commit();
        if(debug == true){
            System.out.println("Query run successfully");
        }
        return true;
    }
    catch(Exception e)
    {
        System.err.println("System Exception in addDepartment");
        System.err.println(e);
        throw e;
    }
}

```

```

/**
 * This method is used to add a new division to a department
 * in the database
 *
 * @param      division struct that contains all applicable
 *             information
 *
 * @return     boolean
 * @exception  system exception to catch all problems
 */
public boolean addDivision( Navy.divisionStruct divisionData)
                               throws Exception
{
    try
    {
        pstmt = con.prepareStatement(
            " INSERT INTO DIVISION( COMMAND_ID, DEPARTMENT_ID, "+
            " DIVISION_ID, TITLE , "+
            " DIVISION_OFFICER_SSN, DIVISION_CHIEF_SSN, "+
            " TELEPHONE, EMAIL ) "+
            " VALUES ( ? , ? , ? , ? , ? , ? , ? , ? ) ");

        pstmt.setString( 1, divisionData.commandId);
        pstmt.setString( 2, divisionData.departmentId);
        pstmt.setString( 3, divisionData.divisionId);
        pstmt.setString( 4, divisionData.title);
        pstmt.setString( 5, divisionData.headSsn);
        pstmt.setString( 6, divisionData.chiefSsn);
        pstmt.setString( 7, divisionData.telephone);
        pstmt.setString( 8, divisionData.email);
        pstmt.executeUpdate();
        pstmt.close();
        con.commit();
        if(debug == true){
            System.out.println("Query run successfully");
        }
        return true;
    }
    catch(Exception e)
    {
        System.err.println("System Exception in addDivision");
        System.err.println(e);
        throw e;
    }
}

```

```

/**
 * This method is used to add maintenance completed to
 * the database
 *
 * @param      maintenance struct that contains all applicable
 *              information
 *
 * @return      boolean
 * @exception   system exception to catch all problems
 */
public boolean addMaintLog(Navy.maintenanceStruct maintenanceData)
                                throws Exception
{
    java.sql.Date dateConduct =
        java.sql.Date.valueOf(maintenanceData.dateOfConduct);

    try
    {
        double hours =
            (Double.valueOf(maintenanceData.hoursOnJob)).doubleValue();
        pstmt = con.prepareStatement(
            " INSERT INTO MAINTENANCE( PERSON, "+
            " MAINTENANCE_NUMBER, ITEM_ID, PMS_ITEM_NUMBER , "+
            " DATE_CONDUCTED, REPAIR_PRIORITY, "+
            " EMERGENCY_DESCRIPTION, COMMAND_ID , "+
            " DEPARTMENT_ID, DIVISION_ID, HOURS__ON_JOB) "+
            " VALUES ( ? , ? , ? , ? , ? , ? , ? , ? , ? , ? , ? , ? ) ");

        pstmt.setString( 1, maintenanceData.person);
        pstmt.setString( 2, maintenanceData.maintenanceNumber);
        pstmt.setString( 3, maintenanceData.itemId);
        pstmt.setString( 4, maintenanceData.pmsNumber);
        pstmt.setDate( 5, dateConduct);
        pstmt.setString( 6, maintenanceData.priority);
        pstmt.setString( 7, maintenanceData.emDescription);
        pstmt.setString( 8, maintenanceData.commandId);
        pstmt.setString( 9, maintenanceData.departmentId);
        pstmt.setString( 10, maintenanceData.divisionId);
        pstmt.setDouble( 11, hours);
        pstmt.executeUpdate();
        pstmt.close();
        con.commit();
        if(debug == true){
            System.out.println("Query run successfully");
        }
        return true;
    }
    catch(Exception e)
    {
        System.err.println("System Exception in addMaintLog");
        System.err.println(e);
    }
}

```

```

        throw e;
    }
}

/**
 * This method is used to add a new sailor to
 * the database
 *
 * @param    sailor struct that contains all applicable
 *           information
 *
 * @return    boolean
 * @exception system exception to catch all problems
 */
public boolean addSailor( Navy.sailorStruct sailorData)
                        throws Exception
{
    java.util.Date dateNow = new java.util.Date();
    java.sql.Date dateRank =
        java.sql.Date.valueOf(sailorData.dateOfRank);
    java.sql.Date dateBirth =
        java.sql.Date.valueOf(sailorData.dateOfBirth);
    java.sql.Date dateService =
        java.sql.Date.valueOf(sailorData.dateOfService);
    java.sql.Date dateUpdate = new java.sql.Date(dateNow.getYear(),
        dateNow.getMonth(), dateNow.getDay());
    try
    {
        pstmt = con.prepareStatement(
            " INSERT INTO SAILOR( SOCIAL_SECURITY_NUMBER ,"+
            " FIRSTNAME, MIDDLENAME, LASTNAME, "+
            " SEX, RANK_RATE, DATE_OF_RANK, WARFARE_DEVICE_1, "+
            " WARFARE_DEVICE_2, WARFARE_DEVICE_3, "+
            " PRIMARY_NEC_DESIGNATOR, SECONDARY_NEC, "+
            " TERTIARY_NEC, ADDRESS,CITY, STATEORPROVINCE, "+
            " COUNTRY, POSTALCODE, HOMEPHONE, BIRTHDATE, "+
            " SERVICE_ENTRY_DATE, HOMETOWN, HOMESTATE, "+
            " LAST_UPDATED ) VALUES "+
            "( ? , ? , ? , ? , ? , ? , ? , ? , ? , ? , ? , ? , ? , ? , "+
            " ? , ? , ? , ? , ? , ? , ? , ? , ? , ? , ? , ? ) ";

        pstmt.setString( 1, sailorData.ssn);
        pstmt.setString( 2, sailorData.firstName);
        pstmt.setString( 3, sailorData.middleName);
        pstmt.setString( 4, sailorData.lastName);
        pstmt.setString( 5, sailorData.sex);
        pstmt.setString( 6, sailorData.rankRate);
        pstmt.setDate( 7, dateRank);
        pstmt.setString( 8, sailorData.device1);
    }
}

```

```

        pstmt.setString( 9, sailorData.device2);
        pstmt.setString( 10, sailorData.device3);
        pstmt.setString( 11, sailorData.primaryNec);
        pstmt.setString( 12, sailorData.secondaryNec);
        pstmt.setString( 13, sailorData.tertiaryNec);
        pstmt.setString( 14, sailorData.address);
        pstmt.setString( 15, sailorData.city);
        pstmt.setString( 16, sailorData.state);
        pstmt.setString( 17, sailorData.country);
        pstmt.setString( 18, sailorData.postalCode);
        pstmt.setString( 19, sailorData.homePhone);
        pstmt.setDate( 20, dateBirth);
        pstmt.setDate( 21, dateService);
        pstmt.setString( 22, sailorData.homeTown);
        pstmt.setString( 23, sailorData.homeState);
        pstmt.setDate( 24, dateUpdate);
        pstmt.executeUpdate();
        pstmt.close();
        con.commit();
        if(debug == true){
            System.out.println("Query run successfully");
        }
        return true;
    }
    catch(Exception e)
    { System.err.println("System Exception in addSailor");
      System.err.println(e);
      throw e;
    }
}

/**
 * This method is used to add a new sea mission to
 * the database
 *
 * @param      seamission struct that contains all applicable
 *              information
 *
 * @return     boolean
 *
 * @exception  system exception to catch all problems
 */
public boolean addSeaMission( Navy.seamissionStruct seamissionData)
    throws Exception
{
    java.sql.Date dateMission =
        java.sql.Date.valueOf(seamissionData.dateOfMission);

    try
    {
        pstmt = con.prepareStatement(

```



```

        " INSERT INTO SEA_MISSION( SEA_MISSION_NUMBER, "+
        " DATE_OF_MISSION, TYPE_OF_MISSION, EXERCISE_NAME , "+
        " PLATFORM_MODE, PLATFORM_TYPE, "+
        " PLATFORM_NUMBER, PLATFORM_NAME , "+
        " MISSION_AREA_ID) "+
        " VALUES ( ? , ? , ? , ? , ? , ? , ? , ? , ? , ? ) ";

    pstmt.setString( 1, seamissionData.missionNumber);
    pstmt.setDate( 2, dateMission);
    pstmt.setString( 3, seamissionData.typeOfMission);
    pstmt.setString( 4, seamissionData.exerciseName);
    pstmt.setString( 5, seamissionData.platformMode);
    pstmt.setString( 6, seamissionData.platformType);
    pstmt.setString( 7, seamissionData.platformNumber);
    pstmt.setString( 8, seamissionData.platformName);
    pstmt.setString( 9, seamissionData.missionArea);
    pstmt.executeUpdate();
    pstmt.close();
    con.commit();
    if(debug == true){
        System.out.println("Query run successfully");
    }
    return true;
}
catch(Exception e)
{ System.err.println("System Exception in addSeaMission");
  System.err.println(e);
  throw e;
}
}

/**
 * This method is used to add a new air mission to
 * the database
 *
 * @param      airmission struct that contains all applicable
 *             information
 *
 * @return     boolean
 * @exception  system exception to catch all problems
 */
public boolean addAirMission( Navy.airmissionStruct airmissionData)
                                throws Exception
{

    java.sql.Timestamp timeTakeOff =
        java.sql.Timestamp.valueOf(airmissionData.takeOffTime);
    java.sql.Timestamp timeLanding =
        java.sql.Timestamp.valueOf(airmissionData.landingTime);

```

```

try
{
    double durationTime =
        (Double.valueOf(airmissionData.duration)).doubleValue();

    pstmt = con.prepareStatement(
        " INSERT INTO FLIGHT_MISSION( FLIGHT_MISSION_NUMBER, "+
        " DATE_TIME_OF_TAKE_OFF, "+
        " DATE_TIME_OF_LANDING, DURATION, PLATFORM_TYPE, "+
        " PLATFORM_NUMBER, SQUADRON, SQUADRON_CREW, "+
        " STAGED_FROM_BASE_, STAGED_FROM_CITY_, "+
        " STAGED_FROM_COUNTRY_, TYPE_OF_MISSION, "+
        " EXERCISE_NAME, "+
        " MISSION_AREA_ID, TRACK_ID ) "+
        " VALUES ( ? , ? , ? , ? , ? , ? , ? , ? , ? , "+
        " ? , ? , ? , ? , ? , ? , ? , ? ) ";

    pstmt.setString( 1, airmissionData.missionNumber);
    pstmt.setTimestamp( 2, timeTakeOff);
    pstmt.setTimestamp( 3, timeLanding);
    pstmt.setDouble( 4, durationTime);
    pstmt.setString( 5, airmissionData.platformType);
    pstmt.setString( 6, airmissionData.platformNumber);
    pstmt.setString( 7, airmissionData.squadron);
    pstmt.setString( 8, airmissionData.squadronCrew);
    pstmt.setString( 9, airmissionData.base);
    pstmt.setString( 10, airmissionData.city);
    pstmt.setString( 11, airmissionData.country);
    pstmt.setString( 12, airmissionData.typeMission);
    pstmt.setString( 13, airmissionData.exerciseName);
    pstmt.setString( 14, airmissionData.missionArea);
    pstmt.setString( 15, airmissionData.trackId);
    pstmt.executeUpdate();
    pstmt.close();
    con.commit();
    if(debug == true){

        System.out.println("Query run successfully");
    }
    return true;
}
catch(Exception e)
{
    System.err.println("System Exception in addAirMission");
    System.err.println(e);
    throw e;
}
}

```

```

/**
 * This method is used to modify department information in
 * the database
 *
 * @param      department struct that contains all applicable
 *             information
 *
 * @return     boolean
 * @exception  system exception to catch all problems
 */
public boolean changeDeptInfo( Navy.departmentStruct departmentData)
                                throws Exception
{
    try
    {
        pstmt = con.prepareStatement(
            "UPDATE DEPARTMENT SET "+
            "TITLE = ? , "+
            "TELEPHONE = ? , "+
            "EMAIL = ? , "+
            "DEPARTMENT_HEAD_SSN = ? , "+
            "DEPARTMENT_CHIEF_SSN = ? "+
            "WHERE DEPARTMENT_ID = ? ");

        pstmt.setString( 1, departmentData.title);
        pstmt.setString( 2, departmentData.telephone);
        pstmt.setString( 3, departmentData.email);
        pstmt.setString( 4, departmentData.headSsn);
        pstmt.setString( 5, departmentData.chiefSsn);
        pstmt.setString( 6, departmentData.departmentId);
        pstmt.executeUpdate();
        pstmt.close();
        con.commit();
        if(debug == true){
            System.out.println("Query run successfully");
        }
        return true;
    }
    catch(Exception e)
    {
        System.err.println("System Exception in changeDeptInfo");
        System.err.println(e);
        throw e;
    }
}

```

```

/**
 * This method is used to modify division information in
 * the database
 *
 * @param      division struct that contains all applicable
 *             information
 *
 * @return     boolean
 * @exception  system exception to catch all problems
 */
public boolean changeDivisionInfo(Navy.divisionStruct divisionData)
                                throws Exception
{
    try
    {
        pstmt = con.prepareStatement(
            "UPDATE DIVISION SET "+
            "COMMAND_ID = ? , "+
            "DEPARTMENT_ID = ? , "+
            "DIVISION_ID = ? , "+
            "TITLE = ? , "+
            "TELEPHONE = ? , "+
            "EMAIL = ? , "+
            "DIVISION_OFFICER_SSN = ? , "+
            "DIVISION_CHIEF_SSN = ? "+
            "WHERE DIVISION_ID = ? ");

        pstmt.setString( 1, divisionData.commandId);
        pstmt.setString( 2, divisionData.departmentId);
        pstmt.setString( 3, divisionData.divisionId);
        pstmt.setString( 4, divisionData.title);
        pstmt.setString( 5, divisionData.telephone);
        pstmt.setString( 6, divisionData.email);
        pstmt.setString( 7, divisionData.headSsn);
        pstmt.setString( 8, divisionData.chiefSsn);
        pstmt.setString( 9, divisionData.divisionId);

        pstmt.executeUpdate();
        pstmt.close();
        con.commit();
        if(debug == true){
            System.out.println("Query run successfully");
        }
        return true;
    }
    catch(Exception e)
    {
        System.err.println("System Exception in changeDivisionInfo");
        System.err.println(e);
        throw e;
    }
}

```

```

/**
 * This method is used to check the userName and password
 * against a table maintained in the database for authorization
 *
 * @param      Strings containing the userName, and password
 *
 * @return     boolean
 * @exception  none
 */
public boolean getAuthorization(String userName, String password)
{
    boolean returnValue = false;
    String resPassword = null;
    try
    {
        Vector result = new Vector();
        pstmt = con.prepareStatement(
            " SELECT  PASSWORD "+
            " FROM  AUTHORIZATION_TABLE WHERE NAME LIKE ?" );
        pstmt.setString(1,userName);
        pstmt.execute();
        rs = pstmt.getResultSet();
        while( rs.next()){
            resPassword = rs.getString(1);
        }
        pstmt.close();
        con.commit();
        if(resPassword.equals(password)){
            returnValue = true;
        }
        return returnValue;
    }
    catch( Exception e)
    {
        System.err.println("System Exception in getAuthorization");
        System.err.println(e);
        return false;
    }
}

/**
 * This method is used to return the family member data of a
 * specific sailor
 *
 * @param      Strings containing the sailors ssn, and last name
 *
 * @return     family member info
 * @exception  system exception to catch all problems
 */
public Vector getFamilyMembers(String sailorSsn,
                                String sailorLastName)

```

```

throws Exception

{
try
{
    Vector result = new Vector();
    pstmt = con.prepareStatement(
        " SELECT  SAILOR.RANK_RATE , SAILOR.FIRSTNAME,"+
        " SAILOR.LASTNAME, "+
        " FAMILY_MEMBERS.FAMILY_MEMBER_SOCIAL_SECURITY_,"+
        " FAMILY_MEMBERS.NAME, "+
        " CONVERT(CHAR(12),FAMILY_MEMBERS.BIRTHDATE,107) AS "+
        " DATEX,"+
        " FAMILY_MEMBERS.SEX, "+
        " FAMILY_MEMBERS.SAILOR_SOCIAL_SECURITY_NUMBER "+
        " FROM  SAILOR INNER JOIN FAMILY_MEMBERS ON "+
        " SAILOR.SOCIAL_SECURITY_NUMBER = "+
        " FAMILY_MEMBERS.SAILOR_SOCIAL_SECURITY_NUMBER WHERE "+
        " (SAILOR.LASTNAME LIKE ? OR "+
        " FAMILY_MEMBERS.SAILOR_SOCIAL_SECURITY_NUMBER LIKE"+
        " ? ) ORDER BY FAMILY_MEMBERS.BIRTHDATE");

    pstmt.setString( 1, sailorLastName);
    pstmt.setString( 2, sailorSsn);
    pstmt.execute();
    rs = pstmt.getResultSet();
    while(rs.next())
    {
        Navy.familydescStruct family = new Navy.familydescStruct();
        family.rank = checkString(rs.getString(1));
        family.firstName = checkString(rs.getString(2));
        family.lastName = checkString(rs.getString(3));
        family.memberSsn = checkString(rs.getString(4));
        family.memberName = checkString(rs.getString(5));
        family.memberBirth = checkString(rs.getString(6));
        family.memberSex = checkString(rs.getString(7));
        family.ememberSailorSsn = checkString(rs.getString(8));
        result.addElement(family);
        if(debug == true){
            System.out.println(family.rank + " " + family.firstName + " "
                + family.lastName + " " + family.memberSsn + " " +
                family.memberName + " " + family.memberBirth + " " +
                family.memberSex + " " + family.ememberSailorSsn );
        }
    }
    pstmt.close();
    con.commit();
    return result;
}
catch( Exception e)
{
    System.err.println("System Exception in getFamilyMembers");
    System.err.println(e);
    throw e;
}

```

```

    }
}

/**
 * This method is used to query the database for a specific
 * sailors spouse
 *
 * @param      Strings containing the sailors ssn, and last name
 *
 * @return     spouse info
 * @exception  system exception to catch all problems
 */
public Vector getSpouse(String sailorSsn, String sailorLastName)
    throws Exception
{
    try
    {
        Vector result = new Vector();
        pstmt = con.prepareStatement(
            " SELECT  SAILOR.RANK_RATE , SAILOR.FIRSTNAME, "+
            " SAILOR.LASTNAME, "+
            " SPOUSES.SPOUSE_SOCIAL_SECURITY_NUMBER, "+
            " SPOUSES.SPOUSE_NAME, "+
            " SPOUSES.SPOUSE_DATE_OF_BIRTH, "+
            " SPOUSES.SPOUSE_GENDER, "+
            " SPOUSES.SAILOR_SOCIAL_SECURITY_NUMBER "+
            " FROM  SAILOR INNER JOIN SPOUSES ON "+
            " SAILOR.SOCIAL_SECURITY_NUMBER = "+
            " SPOUSES.SAILOR_SOCIAL_SECURITY_NUMBER  WHERE "+
            " (SAILOR.LASTNAME LIKE ? OR "+
            " SPOUSES.SAILOR_SOCIAL_SECURITY_NUMBER LIKE ? )");

        pstmt.setString( 1, sailorLastName);
        pstmt.setString( 2, sailorSsn);
        pstmt.execute();
        rs = pstmt.getResultSet();
        while( rs.next()){
            Navy.spousedescStruct spouse = new Navy.spousedescStruct();
            spouse.rank = checkString(rs.getString(1));
            spouse.firstName = checkString(rs.getString(2));
            spouse.lastName = checkString(rs.getString(3));
            spouse.spouseSsn = checkString(rs.getString(4));
            spouse.spouseName = checkString(rs.getString(5));
            spouse.spouseBirth = checkString(rs.getString(6));
            spouse.spouseSex = checkString(rs.getString(7));
            spouse.SailorSsn = checkString(rs.getString(8));

            result.addElement(spouse) ;
            if(debug == true){
                System.out.println(spouse.rank + " " + spouse.firstName + " "
                    + spouse.lastName + " " + spouse.spouseSsn +

```

```

        " " + spouse.spouseName + " " +
        spouse.spouseBirth + " " + spouse.spouseSex +
        " " + spouse.SailorSsn);
    }
}
pstmt.close();
con.commit();
return result;
}
catch( Exception e)
{
    System.err.println("System Exception in getSpouse");
    System.err.println(e);
    throw e;
}
}

/**
 * This method is used to query the database for a specific
 * sailors leave data
 *
 * @param      Strings containing the sailors ssn, and last name
 *
 * @return      leave info
 * @exception  system exception to catch all problems
 */
public Vector getLeaveData(String sailorSsn, String sailorLastName)
    throws Exception
{
    try
    {
        Vector result = new Vector();
        pstmt = con.prepareStatement(
            " SELECT  SAILOR.RANK_RATE , SAILOR.FIRSTNAME, "+
            " SAILOR.LASTNAME, "+
            " LEAVE.SOCIAL_SECURITY_NUMBER, "+
            " LEAVE.LEAVE_CONTROL_NUMBER, "+
            " LEAVE.TYPE_OF_LEAVE, LEAVE.DATE_OF_DEPARTURE, "+
            " LEAVE.DATE_OF_RETURN " +
            " FROM  SAILOR INNER JOIN LEAVE ON "+
            " SAILOR.SOCIAL_SECURITY_NUMBER = "+
            " LEAVE.SOCIAL_SECURITY_NUMBER WHERE "+
            " (SAILOR.LASTNAME LIKE ? OR "+
            " SAILOR.SOCIAL_SECURITY_NUMBER LIKE "+
            " ? ) ORDER BY LEAVE.DATE_OF_DEPARTURE");

        pstmt.setString( 1, sailorLastName);
        pstmt.setString( 2, sailorSsn);
        pstmt.execute();
        rs = pstmt.getResultSet();
        while(rs.next())
        {

```



```

        Navy.leavedescStruct leave = new Navy.leavedescStruct();
        leave.rank = checkString(rs.getString(1));
        leave.firstName = checkString(rs.getString(2));
        leave.lastName = checkString(rs.getString(3));
        leave.leaveSsn = checkString(rs.getString(4));
        leave.controlNumber = checkString(rs.getString(5));
        leave.typeOfLeave = checkString(rs.getString(6));
        leave.dateDepart = checkString(rs.getString(7));
        leave.dateReturn = checkString(rs.getString(8));
        result.addElement(leave );
        if(debug == true){
            System.out.println(leave.rank + " " + leave.firstName + " " +
                leave.lastName + " " + leave.leaveSsn + " " +
                leave.controlNumber + " " + leave.typeOfLeave +
                " " + leave.dateDepart + " " + leave.dateReturn);
        }
    }
    pstmt.close();
    con.commit();
    return result;
}
catch( Exception e)
{
    System.err.println("System Exception in getLeaveData");
    System.err.println(e);
    throw e;
}
}

/**
 * This method is used to query the database for a specific
 * sailors PRT data
 *
 * @param      Strings containing the sailors ssn, and last name
 *
 * @return      PRT info
 * @exception  system exception to catch all problems
 */
public Vector getPrtResults(String sailorSsn, String sailorLastName)
    throws Exception
{
    try
    {
        Vector result = new Vector();
        pstmt = con.prepareStatement(
            " SELECT SAILOR.RANK_RATE , "+
            " SAILOR.FIRSTNAME, SAILOR.LASTNAME, "+
            " PRT_DATA.COMMAND_ID, "+
            " PRT_DATA.PRT_IDENTIFIER, "+
            " PRT_DATA.SOCIAL_SECURITY_NUMBER, "+
            " PRT_DATA.HEIGHT ,"+
            " PRT_DATA.WEIGHT, PRT_DATA.PUSHUPS ,"+

```

```

        " PRT_DATA.SITUPS , "+
        " PRT_DATA.RUN_TIME , PRT_DATA.PRT_DATE , "+
        " PRT_DATA.STATUS "+
        " FROM SAILOR INNER JOIN PRT_DATA ON "+
        " SAILOR.SOCIAL_SECURITY_NUMBER = "+
        " PRT_DATA.SOCIAL_SECURITY_NUMBER WHERE "+
        " (SAILOR.LASTNAME LIKE ? OR "+
        " SAILOR.SOCIAL_SECURITY_NUMBER LIKE "+
        " ? ) ORDER BY PRT_DATA.PRT_DATE ");

pstmt.setString( 1, sailorLastName);
pstmt.setString( 2, sailorSsn);
pstmt.execute();
rs = pstmt.getResultSet();
while(rs.next())
{
    Navy.prtdescStruct prt = new Navy.prtdescStruct();
    prt.rank = checkString(rs.getString(1));
    prt.firstName = checkString(rs.getString(2));
    prt.lastName = checkString(rs.getString(3));
    prt.commandId = checkString(rs.getString(4));
    prt.prtId = checkString(rs.getString(5));
    prt.prtSsn = checkString(rs.getString(6));
    prt.height = checkString(rs.getString(7));
    prt.weight = checkString(rs.getString(8));
    prt.pushUps = checkString(rs.getString(9));
    prt.sitUps = checkString(rs.getString(10));
    prt.runTime = checkString(rs.getString(11));
    prt.prtDate = checkString(rs.getString(12));
    prt.status = checkString(rs.getString(13));
    result.addElement(prt );
    if(debug == true){
        System.out.println(prt.rank + " " + prt.firstName + " " +
            prt.lastName + " "+prt.commandId + " " +
            prt.prtId + " " +prt.prtSsn + " " + prt.height +
            " " + prt.weight + " "+ prt.pushUps + " " +
            prt.sitUps + " " +prt.runTime + " "+prt.prtDate +
            " " + prt.status );
    }
}
pstmt.close();
con.commit();
return result;
}
catch( Exception e)
{
    System.err.println("System Exception in getLeaveData");
    System.err.println(e);
    throw e;
}
}

```

```

/**
 * This method is used to query the database for a specific
 * sailors total days deployed
 *
 * @param      Strings containing the sailors ssn, last name,
 *              and start/stop dates
 *
 * @return     total days deployed
 * @exception  system exception to catch all problems
 */
public Vector getDaysDeployed(String sailorSsn, String
                               sailorLastName, String startDate,
                               String stopDate)
                               throws Exception
{

    java.sql.Date dateBegin = java.sql.Date.valueOf(startDate);
    java.sql.Date dateEnd = java.sql.Date.valueOf(stopDate);

    try
    {
        Vector result = new Vector();
        pstmt = con.prepareStatement(
            "SELECT DISTINCT TAD.TANGO_NUMBER, "+
            " TAD.DATE_OF_DEPARTURE, "+
            " SAILOR.RANK_RATE, "+
            " SAILOR.FIRSTNAME, "+
            " SAILOR.LASTNAME, "+
            " COMMAND_INFORMATION.COMMAND_NAME, "+
            " TAD_MASTER.TITLE, "+
            " TAD.DATE_OF_ARRIVAL_AT_DESTINATION, "+
            " TAD.DATE_OF_DEPARTURE_FROM_DESTINA, "+
            " TAD.DESTINATION____COMMAND_ AS COMMAND"+
            " FROM "+
            " TAD_MASTER INNER JOIN TAD INNER JOIN "+
            " SAILOR INNER JOIN BILLET_OCCUPATION "+
            " INNER JOIN COMMAND_INFORMATION ON "+
            " BILLET_OCCUPATION.COMMAND_ID = "+
            " COMMAND_INFORMATION.COMMAND_ID ON "+
            " SAILOR.SOCIAL_SECURITY_NUMBER = "+
            " BILLET_OCCUPATION.SOCIAL_SECURITY_NUMBER ON "+
            " TAD.MEMBER_SOCIAL_SECURITY_NUMBER = "+
            " SAILOR.SOCIAL_SECURITY_NUMBER ON "+
            " TAD_MASTER.CONFERENCE_TRAINING_ID_CODE = "+
            " TAD.TAD_ID_CODE "+
            " WHERE "+
            " (SAILOR.LASTNAME LIKE ? OR "+
            " SAILOR.SOCIAL_SECURITY_NUMBER LIKE ? ) AND "+
            " TAD.DATE_OF_DEPARTURE BETWEEN ? AND ? "+
            " AND TAD.TANGO_NUMBER = TAD.TANGO_NUMBER "+
            " GROUP BY "+
            " TAD.TANGO_NUMBER, "+

```

```

        " SAILOR.RANK_RATE, "+
        " SAILOR.FIRSTNAME, "+
        " SAILOR.LASTNAME, "+
        " COMMAND_INFORMATION.COMMAND_NAME, "+
        " TAD_MASTER.TITLE, "+
        " TAD.DATE_OF_ARRIVAL_AT_DESTINATION, "+
        " TAD.DATE_OF_DEPARTURE_FROM_DESTINA, "+
        " TAD.DESTINATION____COMMAND_, "+
        " TAD.DATE_OF_DEPARTURE "+
        " ORDER BY "+
        " TAD.DATE_OF_DEPARTURE, "+
        " TAD.TANGO_NUMBER ");

pstmt.setString( 1, sailorLastName);
pstmt.setString( 2, sailorSsn);
pstmt.setDate( 3, dateBegin);
pstmt.setDate( 4, dateEnd);
pstmt.execute();
rs = pstmt.getResultSet();
while(rs.next())
{
    Navy.deploymentStruct deploy = new Navy.deploymentStruct();
    deploy.tangoNumber = checkString(rs.getString(1));
    deploy.dateDepart = checkString(rs.getString(2));
    deploy.rankRate = checkString(rs.getString(3));
    deploy.firstName= checkString(rs.getString(4));
    deploy.lastName = checkString(rs.getString(5));
    deploy.commandName = checkString(rs.getString(6));
    deploy.title = checkString(rs.getString(7));
    deploy.dateArrival = checkString(rs.getString(8));
    deploy.dateDeparture = checkString(rs.getString(9));
    deploy.comm = checkString(rs.getString(10));
    result.addElement(deploy);
    if(debug == true){
        System.out.println(deploy.tangoNumber + " "+
            deploy.dateDepart + " " + deploy.rankRate +
            " "+ deploy.firstName + " " + deploy.lastName +
            " " +deploy.commandName + " " + deploy.title +
            " " + deploy.dateArrival + " "+
            deploy.dateDeparture + " "+ deploy.comm);
    }
}
pstmt.close();
con.commit();
return result;
}
catch( Exception e)
{
    System.err.println("System Exception in db getDaysDeployed");
    System.err.println(e);
    e.printStackTrace();
    throw e;
}

```

```
}
```

```
/**
```

```
 * This method is used to query the database for minor property
 * data
 *
 * @param      String containing the command ID
 *
 * @return     property list info
 * @exception  system exception to catch all problems
 */
```

```
public Vector getPropertyList(String commandId) throws Exception
```

```
{
```

```
    try
```

```
    {
```

```
        Vector result = new Vector();
```

```
        pstmt = con.prepareStatement(
```

```
            " SELECT COMMAND_INFORMATION.COMMAND_NAME, "+
```

```
            " PROPERTIES_LISTING.PROPERTY_ID, "+
```

```
            " PROPERTIES_LISTING.PROPERTY_DESCRIPTION, "+
```

```
            " CONVERT(CHAR(12), " AS DATEX, " +
```

```
            " CONVERT(CHAR(12), PROPERTIES_LISTING.PRICE , 107 ) "+
```

```
            " AS PRICE, "+
```

```
            " SAILOR.RANK_RATE, SAILOR.FIRSTNAME, SAILOR.LASTNAME "+
```

```
            " FROM      "+
```

```
            " COMMAND_INFORMATION INNER JOIN "+
```

```
            " PROPERTIES_LISTING INNER JOIN SAILOR ON "+
```

```
            " PROPERTIES_LISTING.SUBCUSTODIED = "+
```

```
            " SAILOR.SOCIAL_SECURITY_NUMBER ON "+
```

```
            " COMMAND_INFORMATION.COMMAND_ID = "+
```

```
            " PROPERTIES_LISTING.COMMAND_ID WHERE "+
```

```
            " PROPERTIES_LISTING.COMMAND_ID = ? "+
```

```
            " AND PROPERTIES_LISTING.DISPOSAL_DATE "+
```

```
            " IS NULL "+
```

```
            " ORDER BY "+
```

```
            " PROPERTIES_LISTING.PROPERTY_ID ");
```

```
        pstmt.setString( 1, commandId);
```

```
        pstmt.execute();
```

```
        rs = pstmt.getResultSet();
```

```
        while(rs.next())
```

```
        {
```

```
            Navy.propertydescStruct propdesc = new Navy.propertydescStruct();
```

```
            propdesc.commandName = checkString(rs.getString(1));
```

```
            propdesc.propertyId = checkString(rs.getString(2));
```

```
            propdesc.propertyDesc = checkString(rs.getString(3));
```

```
            propdesc.dateOfPurchase = checkString(rs.getString(4));
```

```
            propdesc.price = checkString(rs.getString(5));
```

```
            propdesc.rank = checkString(rs.getString(6));
```

```
            propdesc.firstName = checkString(rs.getString(7));
```

```
            propdesc.lastName = checkString(rs.getString(8));
```

```

        result.addElement(propdesc );
        if(debug == true){
            System.out.println(propdesc.commandName + " " +
                propdesc.propertyId +
                " " + propdesc.propertyDesc + " " +
                propdesc.dateOfPurchase +
                " " + propdesc.price + " " +
                propdesc.rank + " " +
                propdesc.firstName +
                " " + propdesc.lastName);
        }
    }
    pstmt.close();
    con.commit();
    return result;
}
catch( Exception e)
{
    System.err.println("System Exception in getPropertyList");
    System.err.println(e);
    throw e;
}
}

/**
 * This method is used to query the database for a specific
 * commands demographic data
 *
 * @param      Strings containing the command ID, and gender desired
 *
 * @return      command demographic info
 * @exception  system exception to catch all problems
 */
public Vector getComDemByGender(String commandId, String sex)
                                throws Exception
{
    try
    {
        Vector result = new Vector();
        pstmt = con.prepareStatement(
            " SELECT COMMAND_INFORMATION.COMMAND_NAME, "+
            " COUNT(*) AS NUMBERX, "+
            " CONVERT(CHAR(12), GETDATE() ,107) AS DATEZ " +
            " FROM SAILOR INNER JOIN BILLET_OCCUPATION "+
            " INNER JOIN COMMAND_INFORMATION ON "+
            " BILLET_OCCUPATION.COMMAND_ID = "+
            " COMMAND_INFORMATION.COMMAND_ID ON "+
            " SAILOR.SOCIAL_SECURITY_NUMBER = "+
            " BILLET_OCCUPATION.SOCIAL_SECURITY_NUMBER "+
            " WHERE (SAILOR.SEX LIKE ? ) AND " +
            " BILLET_OCCUPATION.ACTUAL_DATE_OF_DEPARTURE IS NULL AND " +
            " BILLET_OCCUPATION.COMMAND_ID = ? "+

```

```

        " GROUP BY COMMAND_INFORMATION.COMMAND_NAME ");

    pstmt.setString( 1, sex);
    pstmt.setString( 2, commandId);
    pstmt.execute();
    rs = pstmt.getResultSet();
    while(rs.next())
    {
        Navy.demographicStruct demographic =
            new Navy.demographicStruct();
        demographic.commandName = checkString(rs.getString(1));
        demographic.number = checkString(rs.getString(2));
        demographic.datex = checkString(rs.getString(3));
        result.addElement(demographic);
        if(debug == true){
            System.out.println(demographic.commandName + " " +
                               demographic.number+ " " +
                               demographic.datex );
        }
    }
    pstmt.close();
    con.commit();
    return result;
}
catch( Exception e)
{ System.err.println("System Exception in getComDemByGender");
  System.err.println(e);
  throw e;
}
}

/**
 * This method is used to query the database for a specific
 * commands demographic data by NEC type
 *
 * @param      String containing the command ID, and 3 strings
 *              containing 3 separate NEC types
 *
 * @return     command demographic NEC info
 * @exception  system exception to catch all problems
 */
public Vector getComDemByNec(String commandId, String nec1,
                             String nec2, String nec3)
    throws Exception
{
    try
    {
        Vector result = new Vector();
        pstmt = con.prepareStatement(
            " SELECT COMMAND_INFORMATION.COMMAND_NAME, "+
            " COUNT(*) AS NUMBERX, "+

```

```

        " CONVERT(CHAR(12), GETDATE() ,107) AS DATEZ " +
        " FROM SAILOR INNER JOIN BILLET_OCCUPATION "+
        " INNER JOIN COMMAND_INFORMATION ON"+
        " BILLET_OCCUPATION.COMMAND_ID = "+
        " COMMAND_INFORMATION.COMMAND_ID ON "+
        " SAILOR.SOCIAL_SECURITY_NUMBER = "+
        " BILLET_OCCUPATION.SOCIAL_SECURITY_NUMBER"+
        " WHERE (SAILOR.PRIMARY_NEC_DESIGNATOR LIKE ? OR"+
        " SAILOR.SECONDARY_NEC LIKE ? OR "+
        " SAILOR.TERTIARY_NEC LIKE ? ) AND "+
        " BILLET_OCCUPATION.ACTUAL_DATE_OF_DEPARTURE IS NULL AND "+
        " BILLET_OCCUPATION.COMMAND_ID = ? "+
        " GROUP BY COMMAND_INFORMATION.COMMAND_NAME");

    pstmt.setString( 1, nec1);
    pstmt.setString( 2, nec2);
    pstmt.setString( 3, nec3);
    pstmt.setString( 4, commandId);
    pstmt.execute();
    rs = pstmt.getResultSet();
    while(rs.next())
    {
        Navy.demographicStruct demographic =
            new Navy.demographicStruct();
        demographic.commandName = checkString(rs.getString(1));
        demographic.number = checkString(rs.getString(2));
        demographic.datex = checkString(rs.getString(3));
        result.addElement(demographic);
        if(debug == true){
            System.out.println(demographic.commandName + " " +
                demographic.number+ " " +
                demographic.datex);
        }
    }
    pstmt.close();
    con.commit();
    return result;
}
catch( Exception e)
{
    System.err.println("System Exception in getComDemByGender");
    System.err.println(e);
    throw e;
}
}

```



```

/**
 * This method is used to query the database for a specific
 * commands demographic data based on a specific rank
 *
 * @param      Strings containing the command ID, and paygrade
 *
 * @return      command demographic info
 * @exception   system exception to catch all problems
 */
public Vector getComDemByRank(String commandId, String payGrade)
                                throws Exception
{
    try
    {
        Vector result = new Vector();
        pstmt = con.prepareStatement(
            " SELECT  COMMAND_INFORMATION.COMMAND_NAME, "+"
            " COUNT(*) AS NUMBERX, "+"
            " CONVERT(CHAR(12), GETDATE() ,107) AS DATEZ " +
            " FROM    SAILOR INNER JOIN BILLET_OCCUPATION "+"
            " INNER JOIN COMMAND_INFORMATION ON "+"
            " BILLET_OCCUPATION.COMMAND_ID = "+"
            " COMMAND_INFORMATION.COMMAND_ID ON "+"
            " SAILOR.SOCIAL_SECURITY_NUMBER = "+"
            " BILLET_OCCUPATION.SOCIAL_SECURITY_NUMBER "+"
            " WHERE (SAILOR.PAYGRADE = ? "+"
            " AND BILLET_OCCUPATION.ACTUAL_DATE_OF_DEPARTURE IS "+"
            " NULL AND "+"
            " BILLET_OCCUPATION.COMMAND_ID = ? ) "+"
            " GROUP BY COMMAND_INFORMATION.COMMAND_NAME ");

        pstmt.setString( 1, payGrade);
        pstmt.setString( 2, commandId);
        pstmt.execute();
        rs = pstmt.getResultSet();
        while(rs.next())
        {
            Navy.demographicStruct demographic =
                new Navy.demographicStruct();
            demographic.commandName = checkString(rs.getString(1));
            demographic.number = checkString(rs.getString(2));
            demographic.datex = checkString(rs.getString(3));
            result.addElement(demographic);
            if(debug == true){
                System.out.println(demographic.commandName + " " +
                                    demographic.number+ " " +
                                    demographic.datex );
            }
        }
        pstmt.close();
        con.commit();
        return result;
    }
}

```

```

    }
    catch( Exception e)
    {
        System.err.println("System Exception in getComDemByRank");
        System.err.println(e);
        throw e;
    }
}

/**
 * This method is used to query the database for a specific
 * commands emergency data
 *
 * @param      String containing the command ID
 *
 * @return      command emergency info
 * @exception  system exception to catch all problems
 */
public Vector getEmergencyList(String commandId) throws Exception
{
    try
    {
        Vector result = new Vector();
        pstmt = con.prepareStatement(
            " SELECT COMMAND_INFORMATION.COMMAND_NAME, "+
            " SUBSTRING (DEPARTMENT.DEPARTMENT_ID , 1, 2 ) AS "+
            " DEPARTMENT_ID , "+
            " SAILOR.RANK_RATE, " +
            " SAILOR.FIRSTNAME, "+
            " SAILOR.LASTNAME, "+
            " SAILOR.HOMEPHONE "+
            " FROM COMMAND_INFORMATION INNER JOIN "+
            " DEPARTMENT INNER JOIN SAILOR ON "+
            " DEPARTMENT.DEPARTMENT_HEAD_SSN = "+
            " SAILOR.SOCIAL_SECURITY_NUMBER ON "+
            " COMMAND_INFORMATION.COMMAND_ID = "+
            " DEPARTMENT.COMMAND_ID "+
            " WHERE DEPARTMENT.COMMAND_ID = ? "+
            " GROUP BY DEPARTMENT.DEPARTMENT_ID, "+
            " COMMAND_INFORMATION.COMMAND_NAME, "+
            " SAILOR.RANK_RATE, SAILOR.FIRSTNAME, "+
            " SAILOR.LASTNAME, SAILOR.HOMEPHONE "+
            " UNION "+
            " SELECT  COMMAND_INFORMATION.COMMAND_NAME, "+
            " SUBSTRING (DEPARTMENT.DEPARTMENT_ID , 1, 2 )" +
            " AS DEPARTMENT_ID , "+
            " SAILOR.RANK_RATE, " +
            " SAILOR.FIRSTNAME, "+
            " SAILOR.LASTNAME, "+
            " SAILOR.HOMEPHONE "+
            " FROM  COMMAND_INFORMATION INNER JOIN "+
            " DEPARTMENT INNER JOIN SAILOR ON "+

```

```

        " DEPARTMENT.DEPARTMENT_CHIEF_SSN = " +
        " SAILOR.SOCIAL_SECURITY_NUMBER ON "+
        " COMMAND_INFORMATION.COMMAND_ID = "+
        " DEPARTMENT.COMMAND_ID"+
        " WHERE DEPARTMENT.COMMAND_ID = ? "+
        " GROUP BY "+
        " DEPARTMENT.DEPARTMENT_ID, "+
        " COMMAND_INFORMATION.COMMAND_NAME, "+
        " SAILOR.RANK_RATE, SAILOR.FIRSTNAME, SAILOR.LASTNAME, " +
        " SAILOR.HOMEPHONE ");
pstmt.setString( 1, commandId);
pstmt.setString( 2, commandId);
pstmt.execute();
rs = pstmt.getResultSet();
while(rs.next())
{
    Navy.emergencyStruct emergency = new Navy.emergencyStruct();
    emergency.commandName = checkString(rs.getString(1));
    emergency.deptId = checkString(rs.getString(2));
    emergency.rank = checkString(rs.getString(3));
    emergency.firstName = checkString(rs.getString(4));
    emergency.lastName = checkString(rs.getString(5));
    emergency.homePhone = checkString(rs.getString(6));

    result.addElement(emergency);
    if(debug == true){
        System.out.println(emergency.commandName + " "+
                           emergency.deptId+ " " + emergency.rank + " "+
                           emergency.firstName + " "+ emergency.lastName +
                           " "+ emergency.homePhone );
    }
}
pstmt.close();
con.commit();
return result;
}
catch( Exception e)
{
    System.err.println("System Exception in getEmergencyList");
    System.err.println(e);
    throw e;
}
}

```

```

/**
 * This method is used to query the database for data regarding
 * a specific seamission based on platform type, type of mission,
 * and mission area
 *
 * @param      Strings containing the platform type, type of
 *              mission, area, and start/stop dates
 *
 * @return     seamission data
 * @exception  system exception to catch all problems
 */
public Vector getSeaMissions(String platform, String typeMission,
                             String missionArea, String startDate,
                             String stopDate)
    throws Exception
{
    java.sql.Date dateBegin = java.sql.Date.valueOf(startDate);
    java.sql.Date dateEnd = java.sql.Date.valueOf(stopDate);

    try
    {
        Vector result = new Vector();
        pstmt = con.prepareStatement(
            " SELECT SEA_MISSION_NUMBER, "+
            " CONVERT (CHAR (12), SEA_MISSION.DATE_OF_MISSION, 107) "+
            " AS DATEX, "+
            " TYPE_OF_MISSION, EXERCISE_NAME, "+
            " PLATFORM_MODE, PLATFORM_TYPE, "+
            " PLATFORM_NUMBER, PLATFORM_NAME, "+
            " SEA_MISSION.MISSION_AREA_ID , DESCRIPTION "+
            " FROM SEA_MISSION INNER JOIN MISSION_AREA_DESCRIPTION ON  "+
            " SEA_MISSION.MISSION_AREA_ID = "+
            " MISSION_AREA_DESCRIPTION.MISSION_AREA_ID "+
            " WHERE PLATFORM_TYPE LIKE ? AND "+
            " TYPE_OF_MISSION LIKE ? AND "+
            " SEA_MISSION.MISSION_AREA_ID LIKE ? AND "+
            " DATE_OF_MISSION >= ? AND "+
            " DATE_OF_MISSION < ? "+
            " ORDER BY SEA_MISSION.DATE_OF_MISSION ");

        pstmt.setString( 1, platform);
        pstmt.setString( 2, typeMission);
        pstmt.setString( 3, missionArea);
        pstmt.setDate( 4, dateBegin);
        pstmt.setDate( 5, dateEnd);
        pstmt.execute();
        rs = pstmt.getResultSet();
        while(rs.next())
        {
            Navy.seamissiondescStruct seamission =
                new Navy.seamissiondescStruct();

```

```

        seamission.missionNumber = checkString(rs.getString(1));
        seamission.dateOfMission = checkString(rs.getString(2));
        seamission.typeOfMission= checkString(rs.getString(3));
        seamission.exerName = checkString(rs.getString(4));
        seamission.platformMode = checkString(rs.getString(5));
        seamission.platformType = checkString(rs.getString(6));
        seamission.platformNumber = checkString(rs.getString(7));
        seamission.platformName = checkString(rs.getString(8));
        seamission.missionAreaId = checkString(rs.getString(9));
        seamission.description = checkString(rs.getString(10));

        result.addElement(seamission) ;

        if(debug == true){
            System.out.println(seamission.missionNumber+ " " +
                               seamission.dateOfMission +
                               " " + seamission.typeOfMission+
                               " " + seamission.exerName + " " +
                               seamission.platformMode + " " +
                               seamission.platformType + " " +
                               seamission.platformNumber + " " +
                               seamission.platformName + " " +
                               seamission.missionAreaId + " " +
                               seamission.description) ;
        }
    }
    pstmt.close();
    con.commit();
    return result;
}
catch( Exception e)
{
    System.err.println("System Exception in getSeaMissions");
    System.err.println(e);
    throw e;
}
}

/**
 * This method is used to query the database for data regarding
 * seamissions based on platform type
 *
 * @param      Strings containing the platform type, and start/stop
 *              dates
 *
 * @return      seamission data
 * @exception   system exception to catch all problems
 */
public Vector getSeaMissionsByPlatform(String platform,
                                       String startDate,
                                       String stopDate)
    throws Exception

```

```

{
    java.sql.Date dateBegin = java.sql.Date.valueOf(startDate);
    java.sql.Date dateEnd = java.sql.Date.valueOf(stopDate);
    try
    {
        Vector result = new Vector();
        pstmt = con.prepareStatement(
            " SELECT SEA_MISSION_NUMBER, "+
            " CONVERT(CHAR(12), SEA_MISSION.DATE_OF_MISSION, 107) AS "+
            " DATEX, "+
            " TYPE_OF_MISSION, EXERCISE_NAME, "+
            " PLATFORM_MODE, PLATFORM_TYPE, "+
            " PLATFORM_NUMBER, PLATFORM_NAME, "+
            " SEA_MISSION.MISSION_AREA_ID , DESCRIPTION "+
            " FROM SEA_MISSION INNER JOIN MISSION_AREA_DESCRIPTION ON "+
            " SEA_MISSION.MISSION_AREA_ID = "+
            " MISSION_AREA_DESCRIPTION.MISSION_AREA_ID "+
            " WHERE PLATFORM_TYPE LIKE ? AND "+
            " DATE_OF_MISSION >= ? AND "+
            " DATE_OF_MISSION < ? "+
            " ORDER BY PLATFORM_NUMBER, DATE_OF_MISSION ");
        pstmt.setString( 1, platform);
        pstmt.setDate( 2, dateBegin);
        pstmt.setDate( 3, dateEnd);
        pstmt.execute();
        rs = pstmt.getResultSet();
        while(rs.next())
        {
            Navy.seamissiondescStruct seamission =
                new Navy.seamissiondescStruct();
            seamission.missionNumber = checkString(rs.getString(1));
            seamission.dateOfMission = checkString(rs.getString(2));
            seamission.typeOfMission = checkString(rs.getString(3));
            seamission.exerName = checkString(rs.getString(4));
            seamission.platformMode = checkString(rs.getString(5));
            seamission.platformType = checkString(rs.getString(6));
            seamission.platformNumber = checkString(rs.getString(7));
            seamission.platformName = checkString(rs.getString(8));
            seamission.missionAreaId = checkString(rs.getString(9));
            seamission.description = checkString(rs.getString(10));

            result.addElement(seamission) ;
            if(debug == true){
                System.out.println(seamission.missionNumber+ " " +
                    seamission.dateOfMission + " " +
                    seamission.typeOfMission+
                    " " + seamission.exerName +
                    " " + seamission.platformMode + " " +
                    seamission.platformType + " " +
                    seamission.platformNumber + " " +
                    seamission.platformName + " " +

```

```

        seamission.missionAreaId +" "+
        seamission.description);
    }
}
pstmt.close();
con.commit();
return result;
}
catch( Exception e)
{
    System.err.println("System Exception in
                        getSeaMissionsByPlatform");
    System.err.println(e);
    throw e;
}
}

/**
 * This method is used to query the database for data regarding
 * a specific seamission based on area data
 *
 * @param      Strings containing the mission area, and start/stop
 *              dates
 *
 * @return      seamission area data
 * @exception  system exception to catch all problems
 */
public Vector getSeaMissionsByArea(String missionArea,
                                   String startDate,
                                   String stopDate) throws Exception
{
    java.sql.Date dateBegin = java.sql.Date.valueOf(startDate);
    java.sql.Date dateEnd = java.sql.Date.valueOf(stopDate);

    try
    {
        Vector result = new Vector();
        pstmt = con.prepareStatement(
            " SELECT SEA_MISSION_NUMBER, "+
            " CONVERT(CHAR(12), SEA_MISSION.DATE_OF_MISSION, 107) AS " +
            " DATEX, "+
            " TYPE_OF_MISSION, EXERCISE_NAME, "+
            " PLATFORM_MODE, PLATFORM_TYPE , "+
            " PLATFORM_NUMBER, PLATFORM_NAME, "+
            " SEA_MISSION.MISSION_AREA_ID , DESCRIPTION "+
            " FROM SEA_MISSION INNER JOIN MISSION_AREA_DESCRIPTION ON  "+
            " SEA_MISSION.MISSION_AREA_ID = "+
            " MISSION_AREA_DESCRIPTION.MISSION_AREA_ID "+
            " WHERE "+
            " SEA_MISSION.MISSION_AREA_ID LIKE ?    AND "+
            " DATE_OF_MISSION >= ? AND "+

```

```

        " DATE_OF_MISSION < ? " +
        " ORDER BY SEA_MISSION.DATE_OF_MISSION ");

pstmt.setString( 1, missionArea);
pstmt.setDate( 2, dateBegin);
pstmt.setDate( 3, dateEnd);
pstmt.execute();
rs = pstmt.getResultSet();
while(rs.next())
{
    Navy.seamissiondescStruct seamission =
        new Navy.seamissiondescStruct();
    seamission.missionNumber = checkString(rs.getString(1));
    seamission.dateOfMission = checkString(rs.getString(2));
    seamission.typeOfMission= checkString(rs.getString(3));
    seamission.exerName = checkString(rs.getString(4));
    seamission.platformMode = checkString(rs.getString(5));
    seamission.platformType = checkString(rs.getString(6));
    seamission.platformNumber = checkString(rs.getString(7));
    seamission.platformName = checkString(rs.getString(8));
    seamission.missionAreaId = checkString(rs.getString(9));
    seamission.description = checkString(rs.getString(10));

    result.addElement(seamission) ;
    if(debug == true){
        System.out.println(seamission.missionNumber+ " " +
            seamission.dateOfMission + " " +
            seamission.typeOfMission+
            " " + seamission.exerName + " " +
            seamission.platformMode + " " +
            seamission.platformType + " " +
            seamission.platformNumber + " " +
            seamission.platformName + " " +
            seamission.missionAreaId + " " +
            seamission.description) ;
    }
}
pstmt.close();
con.commit();
return result;
}
catch( Exception e)
{
    System.err.println("System Exception in getSeaMissionsByArea");
    System.err.println(e);
    throw e;
}
}

```



```

/**
 * This method is used to query the database for data regarding
 * a specific seamission based on type of mission
 *
 * @param      Strings containing the type of mission, and start/stop
 *              dates
 *
 * @return      seamission data
 * @exception   system exception to catch all problems
 */
public Vector getSeaMissionsByType(String typeMission,
                                   String startDate,
                                   String stopDate) throws Exception
{
    java.sql.Date dateBegin = java.sql.Date.valueOf(startDate);
    java.sql.Date dateEnd = java.sql.Date.valueOf(stopDate);

    try
    {
        Vector result = new Vector();
        pstmt = con.prepareStatement(
            " SELECT SEA_MISSION_NUMBER, "+
            " CONVERT(CHAR(12), SEA_MISSION.DATE_OF_MISSION, 107) AS "+
            " DATEX, "+
            " TYPE_OF_MISSION, EXERCISE_NAME, "+
            " PLATFORM_MODE, PLATFORM_TYPE, "+
            " PLATFORM_NUMBER, PLATFORM_NAME, "+
            " SEA_MISSION.MISSION_AREA_ID , DESCRIPTION "+
            " FROM SEA_MISSION INNER JOIN MISSION_AREA_DESCRIPTION ON "+
            " SEA_MISSION.MISSION_AREA_ID = "+
            " MISSION_AREA_DESCRIPTION.MISSION_AREA_ID "+
            " WHERE "+
            " TYPE_OF_MISSION LIKE ? AND "+
            " DATE_OF_MISSION >= ? AND "+
            " DATE_OF_MISSION < ? "+
            " ORDER BY SEA_MISSION.DATE_OF_MISSION ");

        pstmt.setString( 1, typeMission);
        pstmt.setDate( 2, dateBegin);
        pstmt.setDate(3, dateEnd);
        pstmt.execute();
        rs = pstmt.getResultSet();
        while(rs.next())
        {
            Navy.seamissiondescStruct seamission =
                new Navy.seamissiondescStruct();
            seamission.missionNumber = checkString(rs.getString(1));
            seamission.dateOfMission = checkString(rs.getString(2));
            seamission.typeOfMission = checkString(rs.getString(3));
            seamission.exerName = checkString(rs.getString(4));
            seamission.platformMode = checkString(rs.getString(5));
        }
    }
}

```

```

        seamission.platformType = checkString(rs.getString(6));
        seamission.platformNumber = checkString(rs.getString(7));
        seamission.platformName = checkString(rs.getString(8));
        seamission.missionAreaId = checkString(rs.getString(9));
        seamission.description = checkString(rs.getString(10));

        result.addElement(seamission) ;
        if(debug == true){
            System.out.println(seamission.missionNumber+ " " +
                                seamission.dateOfMission + " " +
                                seamission.typeOfMission+ " " +
                                seamission.exerName + " " +
                                seamission.platformMode + " " +
                                seamission.platformType + " " +
                                seamission.platformNumber + " " +
                                seamission.platformName + " " +
                                seamission.missionAreaId + " " +
                                seamission.description) ;
        }
    }
    pstmt.close();
    con.commit();
    return result;
}
catch( Exception e)
{
    System.err.println("System Exception in getSeaMissionsByType");
    System.err.println(e);
    throw e;
}
}

/**
 * This method is used to query the database for data regarding
 * a specific seamission based on exercise type
 *
 * @param      Strings containing the exercise name, and start/stop
 *              dates
 *
 * @return      seamission exercise data
 * @exception   system exception to catch all problems
 */
public Vector getSeaMissionsByExercise(String exerciseName,
                                       String startDate,
                                       String stopDate)
    throws Exception
{
    java.sql.Date dateBegin = java.sql.Date.valueOf(startDate);
    java.sql.Date dateEnd = java.sql.Date.valueOf(stopDate);

    try

```

```

{
    Vector result = new Vector();
    pstmt = con.prepareStatement(
        " SELECT SEA_MISSION_NUMBER, "+
        " CONVERT(CHAR(12), SEA_MISSION.DATE_OF_MISSION, 107) AS "+
        " DATEX, "+
        " TYPE_OF_MISSION, EXERCISE_NAME, "+
        " PLATFORM_MODE, PLATFORM_TYPE, "+
        " PLATFORM_NUMBER, PLATFORM_NAME, "+
        " SEA_MISSION.MISSION_AREA_ID , DESCRIPTION "+
        " FROM SEA_MISSION INNER JOIN MISSION_AREA_DESCRIPTION ON "+
        " SEA_MISSION.MISSION_AREA_ID = "+
        " MISSION_AREA_DESCRIPTION.MISSION_AREA_ID "+
        " WHERE "+
        " TYPE_OF_MISSION LIKE 'EXERCISE' AND "+
        " EXERCISE_NAME LIKE ? AND "+
        " DATE_OF_MISSION >= ? AND "+
        " DATE_OF_MISSION < ? "+
        " ORDER BY EXERCISE_NAME, DATE_OF_MISSION ");

    pstmt.setString( 1, exerciseName);
    pstmt.setDate( 2, dateBegin);
    pstmt.setDate(3, dateEnd);
    pstmt.execute();
    rs = pstmt.getResultSet();
    while(rs.next())
    {
        Navy.seamissiondescStruct seamission =
            new Navy.seamissiondescStruct();
        seamission.missionNumber = checkString(rs.getString(1));
        seamission.dateOfMission = checkString(rs.getString(2));
        seamission.typeOfMission = checkString(rs.getString(3));
        seamission.exerName = checkString(rs.getString(4));
        seamission.platformMode = checkString(rs.getString(5));
        seamission.platformType = checkString(rs.getString(6));
        seamission.platformNumber = checkString(rs.getString(7));
        seamission.platformName = checkString(rs.getString(8));
        seamission.missionAreaId = checkString(rs.getString(9));
        seamission.description = checkString(rs.getString(10));

        result.addElement(seamission) ;
        if(debug == true){
            System.out.println(seamission.missionNumber+ " " +
                seamission.dateOfMission + " " +
                seamission.typeOfMission+ " " +
                seamission.exerName + " " +
                seamission.platformMode + " " +
                seamission.platformType + " " +
                seamission.platformNumber + " " +
                seamission.platformName + " " +
                seamission.missionAreaId + " " +
                seamission.description) ;
        }
    }
}

```

```

    }
    }
    pstmt.close();
    con.commit();
    return result;
}
catch( Exception e)
{
    System.err.println("System Exception in getSeaMissionsByType");
    System.err.println(e);
    throw e;
}
}

/**
 * This method is used to query the database for data regarding
 * a specific seamission based on a platform name
 *
 * @param      Strings containing the platform name, and start/stop
 *              dates
 *
 * @return     platform data
 * @exception  system exception to catch all problems
 */
public Vector getSeaMisByPlatformName(String platform,
                                       String startDate,
                                       String stopDate)
    throws Exception
{
    java.sql.Date dateBegin = java.sql.Date.valueOf(startDate);
    java.sql.Date dateEnd = java.sql.Date.valueOf(stopDate);

    try
    {
        Vector result = new Vector();
        pstmt = con.prepareStatement(
            " SELECT SEA_MISSION_NUMBER, "+
            " CONVERT(CHAR(12), SEA_MISSION.DATE_OF_MISSION, 107) AS "+
            " DATEX, "+
            " TYPE_OF_MISSION, EXERCISE_NAME, "+
            " PLATFORM_MODE, PLATFORM_TYPE , "+
            " PLATFORM_NUMBER, PLATFORM_NAME, "+
            " SEA_MISSION.MISSION_AREA_ID , DESCRIPTION "+
            " FROM SEA_MISSION INNER JOIN MISSION_AREA_DESCRIPTION ON  "+
            " SEA_MISSION.MISSION_AREA_ID = "+
            " MISSION_AREA_DESCRIPTION.MISSION_AREA_ID "+
            " WHERE "+
            " PLATFORM_NAME LIKE ? AND "+
            " DATE_OF_MISSION >= ? AND "+
            " DATE_OF_MISSION < ? "+
            " ORDER BY PLATFORM_NAME, DATE_OF_MISSION ");
    }
}

```

```

pstmt.setString( 1,platform);
pstmt.setDate( 2, dateBegin);
pstmt.setDate(3, dateEnd);
pstmt.execute();
rs = pstmt.getResultSet();
while(rs.next())
{
    Navy.seamissiondescStruct seamission =
        new Navy.seamissiondescStruct();
    seamission.missionNumber = checkString(rs.getString(1));
    seamission.dateOfMission = checkString(rs.getString(2));
    seamission.typeOfMission= checkString(rs.getString(3));
    seamission.exerName = checkString(rs.getString(4));;
    seamission.platformMode = checkString(rs.getString(5));
    seamission.platformType = checkString(rs.getString(6));
    seamission.platformNumber = checkString(rs.getString(7));
    seamission.platformName = checkString(rs.getString(8));
    seamission.missionAreaId = checkString(rs.getString(9));
    seamission.description = checkString(rs.getString(10));

    result.addElement(seamission) ;
    if(debug == true){
        System.out.println(seamission.missionNumber+ " " +
            seamission.dateOfMission + " " +
            seamission.typeOfMission+ " " +
            seamission.exerName + " " +
            seamission.platformMode + " " +
            seamission.platformType + " " +
            seamission.platformNumber + " " +
            seamission.platformName + " " +
            seamission.missionAreaId + " " +
            seamission.description) ;
    }
}
pstmt.close();
con.commit();
return result;
}
catch( Exception e)
{ System.err.println("System Exception in
    getSeaMisByPlatformName");
    System.err.println(e);
    throw e;
}
}

```

```

/**
 * This method is used to query the database for data regarding
 * airmissions
 *
 * @param      Strings containing the squadron name, platform,
 *              type mission, mission area and start/stop dates
 *
 * @return     airmission data
 * @exception  system exception to catch all problems
 */
public Vector getAirMissions(String squadron, String platform,
                             String typeMission, String missionArea,
                             String startDate, String stopDate)
                             throws Exception
{

    java.sql.Date dateBegin = java.sql.Date.valueOf(startDate);
    java.sql.Date dateEnd = java.sql.Date.valueOf(stopDate);

    try
    {
        Vector result = new Vector();
        pstmt = con.prepareStatement(
            " SELECT FLIGHT_MISSION_NUMBER, DATE_TIME_OF_TAKE_OFF, "+
            " DATE_TIME_OF_LANDING,DURATION,  PLATFORM_TYPE, "+
            " PLATFORM_NUMBER, SQUADRON, SQUADRON_CREW, "+
            " STAGED_FROM__BASE_, "+
            " STAGED_FROM__CITY_, STAGED_FROM__COUNTRY_, "+
            " TYPE_OF_MISSION, "+
            " EXERCISE_NAME, FLIGHT_MISSION.MISSION_AREA_ID, TRACK_ID,
            " DESCRIPTION "+
            " FROM FLIGHT_MISSION INNER JOIN MISSION_AREA_DESCRIPTION
            " ON "+
            " FLIGHT_MISSION.MISSION_AREA_ID = "+
            " MISSION_AREA_DESCRIPTION.MISSION_AREA_ID "+
            " WHERE SQUADRON LIKE ? AND "+
            " PLATFORM_TYPE LIKE ? AND "+
            " TYPE_OF_MISSION LIKE ? AND "+
            " FLIGHT_MISSION.MISSION_AREA_ID LIKE ?   AND "+
            " DATE_TIME_OF_TAKE_OFF >= ? AND "+
            " DATE_TIME_OF_TAKE_OFF < ? "+
            " ORDER BY FLIGHT_MISSION.DATE_TIME_OF_TAKE_OFF ");
        pstmt.setString( 1, squadron);
        pstmt.setString( 2, platform);
        pstmt.setString( 3, typeMission);
        pstmt.setString( 4, missionArea);
        pstmt.setDate( 5, dateBegin);
        pstmt.setDate( 6, dateEnd);
        pstmt.execute();
        rs = pstmt.getResultSet();
        while(rs.next())
        {

```

```

Navy.airmissiondescStruct airmission =
    new Navy.airmissiondescStruct();
airmission.missionNumber = checkString(rs.getString(1));
airmission.dateTakeOff = checkString(rs.getString(2));
airmission.dateLanding = checkString(rs.getString(3));
airmission.duration = checkString(rs.getString(4));
airmission.platformType = checkString(rs.getString(5));
airmission.platformNumber = checkString(rs.getString(6));
airmission.squadronId = checkString(rs.getString(7));
airmission.squadronCrew = checkString(rs.getString(8));
airmission.stageBase = checkString(rs.getString(9));
airmission.stageCity = checkString(rs.getString(10));
airmission.stageCountry = checkString(rs.getString(11));
airmission.typeOfMission = checkString(rs.getString(12));
airmission.exerName = checkString(rs.getString(13));
airmission.missionAreaId = checkString(rs.getString(14));
airmission.trackId = checkString(rs.getString(15));
airmission.description = checkString(rs.getString(16));

result.addElement(airmission);
if(debug == true){
    System.out.println(airmission.missionNumber + " " +
        airmission.dateTakeOff + " " +
        airmission.dateLanding + " " +
        airmission.duration + " " +
        airmission.platformType + " " +
        airmission.platformNumber + " " +
        airmission.squadronId + " " +
        airmission.squadronCrew + " " +
        airmission.stageBase + " " +
        airmission.stageCity + " " +
        airmission.stageCountry + " " +
        airmission.typeOfMission + " " +
        airmission.exerName + " " +
        airmission.missionAreaId + " " +
        airmission.trackId + " " +
        airmission.description );
}
}
pstmt.close();
con.commit();
return result;
}
catch( Exception e)
{
    System.err.println("System Exception in getAirMissions");
    System.err.println(e);
    throw e;
}
}

```

```

/**
 * This method is used to query the database for data regarding
 * airmissions by a specific platform number
 *
 * @param      Strings containing the platform number, and start/stop
 *              dates
 *
 * @return     airmission platform number data
 * @exception  system exception to catch all problems
 */
public Vector getAirMisByPlatformNo(String platformNo,
                                    String startDate,
                                    String stopDate)
    throws Exception
{
    java.sql.Date dateBegin = java.sql.Date.valueOf(startDate);
    java.sql.Date dateEnd = java.sql.Date.valueOf(stopDate);

    try
    {
        Vector result = new Vector();
        pstmt = con.prepareStatement(
            " SELECT FLIGHT_MISSION_NUMBER, DATE_TIME_OF_TAKE_OFF, "+
            " DATE_TIME_OF_LANDING,DURATION,  PLATFORM_TYPE, "+
            " PLATFORM_NUMBER, SQUADRON, SQUADRON_CREW, "+
            " STAGED_FROM__BASE_, "+
            " STAGED_FROM__CITY_, STAGED_FROM__COUNTRY_, "+
            " TYPE_OF_MISSION, "+
            " EXERCISE_NAME, FLIGHT_MISSION.MISSION_AREA_ID, TRACK_ID, "+
            " DESCRIPTION "+
            " FROM FLIGHT_MISSION INNER JOIN MISSION_AREA_DESCRIPTION "+
            " ON "+
            " FLIGHT_MISSION.MISSION_AREA_ID = "+
            " MISSION_AREA_DESCRIPTION.MISSION_AREA_ID "+
            " WHERE PLATFORM_NUMBER LIKE ? AND "+
            " DATE_TIME_OF_TAKE_OFF >= ? AND "+
            " DATE_TIME_OF_TAKE_OFF < ? "+
            " ORDER BY FLIGHT_MISSION.DATE_TIME_OF_TAKE_OFF ");
        pstmt.setString( 1, platformNo);
        pstmt.setDate( 2, dateBegin);
        pstmt.setDate( 3, dateEnd);
        pstmt.execute();
        rs = pstmt.getResultSet();
        while(rs.next())
        {
            Navy.airmissiondescStruct airmission =
                new Navy.airmissiondescStruct();
            airmission.missionNumber = checkString(rs.getString(1));
            airmission.dateTakeOff = checkString(rs.getString(2));
            airmission.dateLanding = checkString(rs.getString(3));
            airmission.duration = checkString(rs.getString(4));
        }
    }
}

```



```

airmission.platformType = checkString(rs.getString(5));
airmission.platformNumber = checkString(rs.getString(6));
airmission.squadronId = checkString(rs.getString(7));
airmission.squadronCrew = checkString(rs.getString(8));
airmission.stageBase = checkString(rs.getString(9));
airmission.stageCity = checkString(rs.getString(10));
airmission.stageCountry = checkString(rs.getString(11));
airmission.typeOfMission = checkString(rs.getString(12));
airmission.exerName = checkString(rs.getString(13));
airmission.missionAreaId = checkString(rs.getString(14));
airmission.trackId = checkString(rs.getString(15));
airmission.description = checkString(rs.getString(16));

result.addElement(airmission);
if(debug == true){
    System.out.println(airmission.missionNumber + " " +
        airmission.dateTakeOff + " " +
        airmission.dateLanding + " " +
        airmission.duration + " " +
        airmission.platformType + " " +
        airmission.platformNumber + " " +
        airmission.squadronId + " " +
        airmission.squadronCrew + " " +
        airmission.stageBase + " " +
        airmission.stageCity + " " +
        airmission.stageCountry + " " +
        airmission.typeOfMission + " " +
        airmission.exerName + " " +
        airmission.missionAreaId + " " +
        airmission.trackId + " " +
        airmission.description );
}

}

pstmt.close();
con.commit();
return result;
}

catch( Exception e)
{
    System.err.println("System Exception in getAirMisByPlatform");
    System.err.println(e);
    throw e;
}
}

```

```

/**
 * This method is used to query the database for data regarding
 * airmissions by a specific mission area
 *
 * @param      Strings containing the mission area, and start/stop
 *              dates
 *
 * @return     airmission area data
 * @exception  system exception to catch all problems
 */
public Vector getAirMisByArea(String missionArea, String startDate,
                              String stopDate) throws Exception
{
    java.sql.Date dateBegin = java.sql.Date.valueOf(startDate);
    java.sql.Date dateEnd = java.sql.Date.valueOf(stopDate);

    try
    {
        Vector result = new Vector();
        pstmt = con.prepareStatement(
            " SELECT FLIGHT_MISSION_NUMBER, DATE_TIME_OF_TAKE_OFF, "+
            " DATE_TIME_OF_LANDING, DURATION, PLATFORM_TYPE, "+
            " PLATFORM_NUMBER, SQUADRON, SQUADRON_CREW, "+
            " STAGED_FROM__BASE_, "+
            " STAGED_FROM__CITY_, STAGED_FROM__COUNTRY_, "+
            " TYPE_OF_MISSION, "+
            " EXERCISE_NAME, FLIGHT_MISSION.MISSION_AREA_ID, TRACK_ID, "+
            " DESCRIPTION "+
            " FROM FLIGHT_MISSION INNER JOIN MISSION_AREA_DESCRIPTION "+
            " ON "+
            " FLIGHT_MISSION.MISSION_AREA_ID = "+
            " MISSION_AREA_DESCRIPTION.MISSION_AREA_ID "+
            " WHERE "+
            " FLIGHT_MISSION.MISSION_AREA_ID LIKE ?    AND "+
            " DATE_TIME_OF_TAKE_OFF >= ? AND "+
            " DATE_TIME_OF_TAKE_OFF < ? "+
            " ORDER BY FLIGHT_MISSION.DATE_TIME_OF_TAKE_OFF ");

        pstmt.setString( 1, missionArea);
        pstmt.setDate( 2, dateBegin);
        pstmt.setDate( 3, dateEnd);
        pstmt.execute();
        rs = pstmt.getResultSet();
        while(rs.next())
        {
            Navy.airmissiondescStruct airmission =
                new Navy.airmissiondescStruct();
            airmission.missionNumber = checkString(rs.getString(1));
            airmission.dateTakeOff = checkString(rs.getString(2));
            airmission.dateLanding = checkString(rs.getString(3));
            airmission.duration = checkString(rs.getString(4));
        }
    }
}

```

```

airmission.platformType = checkString(rs.getString(5));
airmission.platformNumber = checkString(rs.getString(6));
airmission.squadronId = checkString(rs.getString(7));
airmission.squadronCrew = checkString(rs.getString(8));
airmission.stageBase = checkString(rs.getString(9));
airmission.stageCity = checkString(rs.getString(10));
airmission.stageCountry = checkString(rs.getString(11));
airmission.typeOfMission = checkString(rs.getString(12));
airmission.exerName = checkString(rs.getString(13));
airmission.missionAreaId = checkString(rs.getString(14));
airmission.trackId = checkString(rs.getString(15));
airmission.description = checkString(rs.getString(16));

result.addElement(airmission);
if(debug == true){
    System.out.println(airmission.missionNumber + " " +
        airmission.dateTakeOff+ " " +
        airmission.dateLanding + " " +
        airmission.duration + " " +
        airmission.platformType + " " +
        airmission.platformNumber + " " +
        airmission.squadronId + " " +
        airmission.squadronCrew + " " +
        airmission.stageBase + " " +
        airmission.stageCity + " " +
        airmission.stageCountry + " " +
        airmission.typeOfMission + " " +
        airmission.exerName + " " +
        airmission.missionAreaId + " " +
        airmission.trackId + " " +
        airmission.description );
}
}
pstmt.close();
con.commit();
return result;
}
catch( Exception e)
{
    System.err.println("System Exception in getAirMissions");
    System.err.println(e);
    throw e;
}
}

```

```

/**
 * This method is used to query the database for data regarding
 * airmissions by a mission type
 *
 * @param      Strings containing the type of mission, and start/stop
 *              dates
 *
 * @return     airmission type data
 * @exception  system exception to catch all problems
 */
public Vector getAirMisByType(String typeOfMission,
                              String startDate, String stopDate)
                              throws Exception
{

    java.sql.Date dateBegin = java.sql.Date.valueOf(startDate);
    java.sql.Date dateEnd = java.sql.Date.valueOf(stopDate);

    try
    {
        Vector result = new Vector();
        pstmt = con.prepareStatement(
            " SELECT FLIGHT_MISSION_NUMBER, DATE_TIME_OF_TAKE_OFF, "+
            " DATE_TIME_OF_LANDING,DURATION,  PLATFORM_TYPE,"+
            " PLATFORM_NUMBER, SQUADRON, SQUADRON_CREW, "+
            " STAGED_FROM__BASE_, "+
            " STAGED_FROM__CITY_, STAGED_FROM__COUNTRY_, "+
            " TYPE_OF_MISSION,"+
            " EXERCISE_NAME, FLIGHT_MISSION.MISSION_AREA_ID, TRACK_ID,"+
            " DESCRIPTION "+
            " FROM FLIGHT_MISSION INNER JOIN MISSION_AREA_DESCRIPTION "+
            " ON "+
            " FLIGHT_MISSION.MISSION_AREA_ID = "+
            " MISSION_AREA_DESCRIPTION.MISSION_AREA_ID"+
            " WHERE "+
            " TYPE_OF_MISSION LIKE ?  AND "+
            " DATE_TIME_OF_TAKE_OFF >= ? AND "+
            " DATE_TIME_OF_TAKE_OFF < ? "+
            " ORDER BY FLIGHT_MISSION.DATE_TIME_OF_TAKE_OFF ");

        pstmt.setString( 1, typeOfMission);
        pstmt.setDate( 2, dateBegin);
        pstmt.setDate( 3, dateEnd);
        pstmt.execute();
        rs = pstmt.getResultSet();
        while(rs.next())
        {
            Navy.airmissiondescStruct airmission =
                new Navy.airmissiondescStruct();
            airmission.missionNumber = checkString(rs.getString(1));
            airmission.dateTakeOff = checkString(rs.getString(2));
            airmission.dateLanding = checkString(rs.getString(3));
        }
    }
}

```

```

airmission.duration = checkString(rs.getString(4));
airmission.platformType = checkString(rs.getString(5));
airmission.platformNumber = checkString(rs.getString(6));
airmission.squadronId = checkString(rs.getString(7));
airmission.squadronCrew = checkString(rs.getString(8));
airmission.stageBase = checkString(rs.getString(9));
airmission.stageCity = checkString(rs.getString(10));
airmission.stageCountry = checkString(rs.getString(11));
airmission.typeOfMission = checkString(rs.getString(12));
airmission.exerName = checkString(rs.getString(13));
airmission.missionAreaId = checkString(rs.getString(14));
airmission.trackId = checkString(rs.getString(15));
airmission.description = checkString(rs.getString(16));

result.addElement(airmission);
if(debug == true){
    System.out.println(airmission.missionNumber + " " +
        airmission.dateTakeOff + " " +
        airmission.dateLanding + " " +
        airmission.duration + " " +
        airmission.platformType + " " +
        airmission.platformNumber + " " +
        airmission.squadronId + " " +
        airmission.squadronCrew + " " +
        airmission.stageBase + " " +
        airmission.stageCity + " " +
        airmission.stageCountry + " " +
        airmission.typeOfMission + " " +
        airmission.exerName + " " +
        airmission.missionAreaId + " " +
        airmission.trackId + " " +
        airmission.description );
}
}
pstmt.close();
con.commit();
return result;
}
catch( Exception e)
{
    System.err.println("System Exception in getAirMissions");
    System.err.println(e);
    throw e;
}
}

```

```

/**
 * This method is used to query the database for data regarding
 * airmissions by exercise
 *
 * @param      Strings containing the exercise name, and start/stop
 *              dates
 *
 * @return      airmission exercise data
 * @exception   system exception to catch all problems
 */
public Vector getAirMisByExercise(String exerciseName,
                                   String startDate,
                                   String stopDate)
    throws Exception
{
    java.sql.Date dateBegin = java.sql.Date.valueOf(startDate);
    java.sql.Date dateEnd = java.sql.Date.valueOf(stopDate);

    try
    {
        Vector result = new Vector();
        pstmt = con.prepareStatement(
            " SELECT FLIGHT_MISSION_NUMBER, DATE_TIME_OF_TAKE_OFF, "+
            " DATE TIME OF LANDING,DURATION,  PLATFORM_TYPE, "+
            " PLATFORM_NUMBER, SQUADRON, SQUADRON_CREW, "+
            " STAGED_FROM_BASE, "+
            " STAGED_FROM_CITY, STAGED_FROM_COUNTRY, "+
            " TYPE_OF_MISSION, "+
            " EXERCISE_NAME, FLIGHT_MISSION.MISSION_AREA_ID, TRACK_ID, "+
            " DESCRIPTION "+
            " FROM FLIGHT_MISSION INNER JOIN MISSION_AREA_DESCRIPTION "+
            " ON "+
            " FLIGHT_MISSION.MISSION_AREA_ID = "+
            " MISSION_AREA_DESCRIPTION.MISSION_AREA_ID "+
            " WHERE "+
            " TYPE_OF_MISSION LIKE 'EXERCISE' AND "+
            " EXERCISE_NAME LIKE ? AND "+
            " DATE_TIME_OF_TAKE_OFF >= ? AND "+
            " DATE_TIME_OF_TAKE_OFF < ? "+
            " ORDER BY FLIGHT_MISSION.DATE_TIME_OF_TAKE_OFF ");

        pstmt.setString( 1, exerciseName);
        pstmt.setDate( 2, dateBegin);
        pstmt.setDate( 3, dateEnd);
        pstmt.execute();
        rs = pstmt.getResultSet();
        while(rs.next())
        {
            Navy.airmissiondescStruct airmission =
                new Navy.airmissiondescStruct();
            airmission.missionNumber = checkString(rs.getString(1));

```

```

airmission.dateTakeOff = checkString(rs.getString(2));
airmission.dateLanding = checkString(rs.getString(3));
airmission.duration = checkString(rs.getString(4));
airmission.platformType = checkString(rs.getString(5));
airmission.platformNumber = checkString(rs.getString(6));
airmission.squadronId = checkString(rs.getString(7));
airmission.squadronCrew = checkString(rs.getString(8));
airmission.stageBase = checkString(rs.getString(9));
airmission.stageCity = checkString(rs.getString(10));
airmission.stageCountry = checkString(rs.getString(11));
airmission.typeOfMission = checkString(rs.getString(12));
airmission.exerName = checkString(rs.getString(13));
airmission.missionAreaId = checkString(rs.getString(14));
airmission.trackId = checkString(rs.getString(15));
airmission.description = checkString(rs.getString(16));

result.addElement(airmission);
if(debug == true){
    System.out.println(airmission.missionNumber + " "+
        airmission.dateTakeOff+ " " +
        airmission.dateLanding +" " +
        airmission.duration + " " +
        airmission.platformType + " " +
        airmission.platformNumber +" " +
        airmission.squadronId + " "+
        airmission.squadronCrew + " " +
        airmission.stageBase + " "+
        airmission.stageCity + " " +
        airmission.stageCountry + " " +
        airmission.typeOfMission + " "+
        airmission.exerName + " " +
        airmission.missionAreaId + " "+
        airmission.trackId + " " +
        airmission.description );
}

}

pstmt.close();
con.commit();
return result;
}

catch( Exception e)
{
    System.err.println("System Exception in getAirMissions");
    System.err.println(e);
    throw e;
}
}

```

```

/**
 * This method is used to query the database for data regarding
 * airmissions by a specific squadron
 *
 * @param      Strings containing the squadron name, and start/stop
 *              dates
 *
 * @return     airmission squadron data
 * @exception  system exception to catch all problems
 */
public Vector getAirMisBySquadron(String squadronName,
                                   String startDate,
                                   String stopDate)
                                   throws Exception
{

    java.sql.Date dateBegin = java.sql.Date.valueOf(startDate);
    java.sql.Date dateEnd = java.sql.Date.valueOf(stopDate);

    try
    {
        Vector result = new Vector();
        pstmt = con.prepareStatement(
            " SELECT FLIGHT_MISSION_NUMBER, DATE_TIME_OF_TAKE_OFF, "+
            " DATE_TIME_OF_LANDING,DURATION,  PLATFORM_TYPE, "+
            " PLATFORM_NUMBER, SQUADRON, SQUADRON_CREW, "+
            " STAGED_FROM__BASE_, "+
            " STAGED_FROM__CITY_, STAGED_FROM__COUNTRY_, "+
            " TYPE_OF_MISSION, "+
            " EXERCISE_NAME, FLIGHT_MISSION.MISSION_AREA_ID, TRACK_ID, "+
            " DESCRIPTION "+
            " FROM FLIGHT_MISSION INNER JOIN MISSION_AREA_DESCRIPTION
            " ON "+
            " FLIGHT_MISSION.MISSION_AREA_ID = "+
            " MISSION_AREA_DESCRIPTION.MISSION_AREA_ID "+
            " WHERE "+
            " SQUADRON LIKE ? AND "+
            " DATE_TIME_OF_TAKE_OFF >= ? AND "+
            " DATE_TIME_OF_TAKE_OFF < ? "+
            " ORDER BY FLIGHT_MISSION.DATE_TIME_OF_TAKE_OFF ");

        pstmt.setString( 1, squadronName);
        pstmt.setDate( 2, dateBegin);
        pstmt.setDate( 3, dateEnd);
        pstmt.execute();
        rs = pstmt.getResultSet();
        while(rs.next())
        {
            Navy.airmissiondescStruct airmission =
                new Navy.airmissiondescStruct();
            airmission.missionNumber = checkString(rs.getString(1));
            airmission.dateTakeOff = checkString(rs.getString(2));

```



```

airmission.dateLanding = checkString(rs.getString(3));
airmission.duration = checkString(rs.getString(4));
airmission.platformType = checkString(rs.getString(5));
airmission.platformNumber = checkString(rs.getString(6));
airmission.squadronId = checkString(rs.getString(7));
airmission.squadronCrew = checkString(rs.getString(8));
airmission.stageBase = checkString(rs.getString(9));
airmission.stageCity = checkString(rs.getString(10));
airmission.stageCountry = checkString(rs.getString(11));
airmission.typeOfMission = checkString(rs.getString(12));
airmission.exerName = checkString(rs.getString(13));
airmission.missionAreaId = checkString(rs.getString(14));
airmission.trackId = checkString(rs.getString(15));
airmission.description = checkString(rs.getString(16));

result.addElement(airmission);
if(debug == true){
    System.out.println(airmission.missionNumber + " " +
        airmission.dateTakeOff+ " " +
        airmission.dateLanding + " " +
        airmission.duration + " " +
        airmission.platformType + " " +
        airmission.platformNumber + " " +
        airmission.squadronId + " " +
        airmission.squadronCrew + " " +
        airmission.stageBase + " " +
        airmission.stageCity + " " +
        airmission.stageCountry + " " +
        airmission.typeOfMission + " " +
        airmission.exerName + " " +
        airmission.missionAreaId + " " +
        airmission.trackId + " " +
        airmission.description );
}

}
pstmt.close();
con.commit();
return result;
}
catch( Exception e)
{
    System.err.println("System Exception in getAirMissions");
    System.err.println(e);
    throw e;
}
}

```

```

/**
 * This method is used to query the database for data regarding
 * maintenance data of a specific command
 *
 * @param      Strings containing the command ID, department ID
 *              and the start/stop dates
 *
 * @return     maintenance data
 * @exception  system exception to catch all problems
 */
public Vector getMaintByCommand(String commandId,
                                String departmentId,
                                String startDate, String stopDate)
                                throws Exception
{
    java.sql.Date dateBegin = java.sql.Date.valueOf(startDate);
    java.sql.Date dateEnd = java.sql.Date.valueOf(stopDate);

    try
    {
        Vector result = new Vector();
        pstmt = con.prepareStatement(
            " SELECT COMMAND_INFORMATION.COMMAND_NAME, "+
            " PMS_DESCRIPTION.DESCRPTION, "+
            " MAINTENANCE.PERSON, MAINTENANCE.ITEM_ID, "+
            " MAINTENANCE.MAINTENANCE_NUMBER, "+
            " CONVERT(CHAR(12), MAINTENANCE.DATE_CONDUCTED, 107) AS "+
            " DATEX, "+
            " MAINTENANCE.REPAIR_PRIORITY , "+
            " MAINTENANCE.EMERGENCY_DESCRIPTION , "+
            " MAINTENANCE.DEPARTMENT_ID, "+
            " MAINTENANCE.HOURS_ON_JOB "+
            " FROM COMMAND_INFORMATION INNER JOIN PMS_DESCRIPTION  "+
            " INNER JOIN MAINTENANCE ON "+
            " MAINTENANCE.PMS_ITEM_NUMBER = "+
            " PMS_DESCRIPTION.PMS_ITEM_NUMBER ON "+
            " COMMAND_INFORMATION.COMMAND_ID = "+
            " MAINTENANCE.COMMAND_ID "+
            " WHERE "+
            " MAINTENANCE.COMMAND_ID = ? AND "+
            " MAINTENANCE.DEPARTMENT_ID LIKE  ? AND "+
            " MAINTENANCE.DATE_CONDUCTED > ?  AND "+
            " MAINTENANCE.DATE_CONDUCTED <= ?  "+
            " ORDER BY MAINTENANCE.DATE_CONDUCTED,  "+
            " MAINTENANCE.MAINTENANCE_NUMBER ");

        pstmt.setString( 1, commandId);
        pstmt.setString( 2, departmentId);
        pstmt.setDate( 3, dateBegin);
        pstmt.setDate( 4, dateEnd);
        pstmt.execute();
    }
}

```

```

rs = pstmt.getResultSet();
while(rs.next())
{
    Navy.comaintenanceStruct maintenance =
        new Navy.comaintenanceStruct();
    maintenance.commandId = checkString(rs.getString(1));
    maintenance.description = checkString(rs.getString(2));
    maintenance.person = checkString(rs.getString(3));
    maintenance.itemId = checkString(rs.getString(4));
    maintenance.maintNo = checkString(rs.getString(5));
    maintenance.dateConduct = checkString(rs.getString(6));
    maintenance.priority = checkString(rs.getString(7));
    maintenance.emergencyDesc = checkString(rs.getString(8));
    maintenance.deptId = checkString(rs.getString(9));
    maintenance.hours = checkString(rs.getString(10));

    result.addElement(maintenance);
    if(debug == true){
        System.out.println(maintenance.commandId + " " +
            maintenance.description + " " +
            maintenance.person + " " +
            maintenance.itemId + " " +
            maintenance.maintNo + " " +
            maintenance.dateConduct + " " +
            maintenance.priority + " " +
            maintenance.emergencyDesc + " " +
            maintenance.deptId + " " +
            maintenance.hours );
    }
}
pstmt.close();
con.commit();
System.out.println("Query run successfully");
return result;
}
catch( Exception e)
{
    System.err.println("System Exception in getMaintByCommand");
    System.err.println(e);
    throw e;
}
}

/**
 * This method is used to query the database for data regarding
 * the maintenance of a specific item
 *
 * @param      String containing the equipment item
 *
 * @return     maintenance data
 * @exception  system exception to catch all problems
 */

```

```

public Vector getMaintByItem(String item) throws Exception
{
    try
    {
        Vector result = new Vector();

        pstmt = con.prepareStatement(
            " SELECT PMS_DESCRIPTION.DESCRPTION, "+
            " PROPERTIES_LISTING.PROPERTY_DESCRIPTION, "+
            " MAINTENANCE.PERSON, "+
            " MAINTENANCE.ITEM_ID , "+
            " MAINTENANCE.MAINTENANCE_NUMBER, "+
            " CONVERT (CHAR (12), MAINTENANCE.DATE_CONDUCTED, 107) AS "+
            " DATEX, "+
            " MAINTENANCE.REPAIR_PRIORITY , "+
            " MAINTENANCE.EMERGENCY_DESCRIPTION , "+
            " MAINTENANCE.DEPARTMENT_ID, "+
            " MAINTENANCE.HOURS_ON_JOB "+
            " FROM PROPERTIES_LISTING INNER JOIN PMS_DESCRIPTION      "+
            " INNER JOIN MAINTENANCE ON "+
            " MAINTENANCE.PMS_ITEM_NUMBER = "+
            " PMS_DESCRIPTION.PMS_ITEM_NUMBER ON "+
            " PROPERTIES_LISTING.PROPERTY_ID = MAINTENANCE.ITEM_ID "+
            " WHERE "+
            " MAINTENANCE.ITEM_ID = ? "+
            " ORDER BY MAINTENANCE.DATE_CONDUCTED ");

        pstmt.setString( 1, item);
        pstmt.execute();
        rs = pstmt.getResultSet();
        while(rs.next())
        {
            Navy.itemmaintenanceStruct queryitem =
                new Navy.itemmaintenanceStruct();
            queryitem.pmsDesc = checkString(rs.getString(1));
            queryitem.description = checkString(rs.getString(2));
            queryitem.person = checkString(rs.getString(3));
            queryitem.itemId = checkString(rs.getString(4));
            queryitem.maintNo = checkString(rs.getString(5));
            queryitem.dateConduct = checkString(rs.getString(6));
            queryitem.priority = checkString(rs.getString(7));
            queryitem.emergencyDesc = checkString(rs.getString(8));
            queryitem.deptId = checkString(rs.getString(9));
            queryitem.hours = checkString(rs.getString(10));

            result.addElement(queryitem);
            if(debug == true){
                System.out.println(queryitem.pmsDesc + " " +
                    queryitem.description+ " " +
                    queryitem.person + " " +
                    queryitem.itemId + " " +

```

```

        queryitem.maintNo + " " +
        queryitem.dateConduct + " " +
        queryitem.priority + " " +
        queryitem.emergencyDesc + " " +
        queryitem.deptId + " " +
        queryitem.hours );
    }
}
pstmt.close();
con.commit();
System.out.println("Query run successfully");
return result;
}
catch( Exception e)
{
    System.err.println("System Exception in getMaintByItem");
    System.err.println(e);
    throw e;
}
}
}
}

} //end DBHelper.java

```

APPENDIX D. CLIENT FILES

This appendix provides several selected excerpts from the source code required for the Client implementation. The client side has a total of 85 classes, therefore for brevity we have included only the most significant classes to include: Gui.java, CorbaBean.java, AdminPanel, CommandPanel, OperationsPanel, MaintenancePanel, BudgetPanel, SwitchPanel, WelcomePanel, AddFamilyBean, ChangeAddressBean, GetDaysDeployedBean, GetDaysDeployedPanel1, and GetDaysDeployedPanel2 classes. In addition, we have included the Gui.html and Guijar.html files.

```
//-----
// Filename      : Gui.java
// Authors       : Murat Akbay & Steve Lewis
// Date          : 10/17/1998
// Compiler      : JDK1.1.6 with Symantec JIT Compiler
//-----

/**
 * This class is the definition for the main applet class that
 * holds a reference to all of the components that are used by
 * the client to invoke operations on the CORBA server
 * @authors Murat Akbay & Steve Lewis
 */

import java.awt.*;
import java.applet.*;
import borland.jbcl.layout.*;
import borland.jbcl.control.*;
import com.sun.java.swing.*;
import java.awt.event.*;

public class Gui extends JApplet implements ActionListener{

    CorbaBean corbaBean;
    AdminPanel adminPanel;
    BudgetPanel budgetPanel;
    CommandPanel commandPanel;
    MaintenancePanel maintenancePanel;
    OperationsPanel operationsPanel;
    WelcomePanel welcomePanel;
    SwitchPanel switchPanel;
    XYLayout xYLayout1 = new XYLayout();
    JMenuBar menubar = new JMenuBar();
    JMenu menu = new JMenu("MENU");
    JMenuItem item1 = new JMenuItem("    Switch Panel    ");
    JMenuItem item2 = new JMenuItem("    Administration    ");
```

```

JMenuItem item3 = new JMenuItem(" Budget & Supply ");
JMenuItem item4 = new JMenuItem(" Command ");
JMenuItem item5 = new JMenuItem (" Maintenance ");
JMenuItem item6 = new JMenuItem("Operations & Trng.");
JMenuItem item7 = new JMenuItem(" Welcome ");

```

```

JPanel displayPanel = new JPanel();
CardLayout cardLayout1 = new CardLayout();
boolean authorized = false;

```

```

/**
 * This method is used by the browser to initialize the applet
 *
 * @param      none
 * @return      void
 * @exception   none
 */
public void init() {
    try{
        jbInit();
    }
    catch (Exception ex) {
        ex.printStackTrace();
    }
}

```

```

/**
 * This method is used by the panels to set the authorization of the
 * client
 *
 * @param      inValue  authorization of the client
 * @return      void
 * @exception   none
 */
void setAuthorization(boolean inValue){
    authorized = inValue;
}

```

```

/**
 * This method is used by init() to initialize the panels
 *
 * @param      none
 * @return      void
 * @exception   default Jbuilder exception
 */
private void jbInit() throws Exception {
    corbaBean = new CorbaBean(this);
    xYLayout1.setHeight(590);
    adminPanel = new AdminPanel(this);
}

```

```

budgetPanel = new BudgetPanel(this);
commandPanel = new CommandPanel(this);
maintenancePanel = new MaintenancePanel(this);
operationsPanel = new OperationsPanel(this);
welcomePanel = new WelcomePanel(this);
switchPanel = new SwitchPanel(this);
item1.addActionListener(this);
item2.addActionListener(this);
item3.addActionListener(this);
item4.addActionListener(this);
item5.addActionListener(this);
item6.addActionListener(this);
item7.addActionListener(this);
xYLayout1.setWidth(790);
menu.add(item1);
menu.add(item2);
menu.add(item3);
menu.add(item4);
menu.add(item5);
menu.add(item6);
menu.add(item7);
menubar.add(menu);
setJMenuBar(menubar);
setBackground(Color.lightGray);
this.getContentPane().setLayout(xYLayout1);
displayPanel.setLayout(cardLayout1);
displayPanel.add("Administration", adminPanel);
displayPanel.add("Budget&Supply", budgetPanel);
displayPanel.add("Command", commandPanel);
displayPanel.add("Maintenance", maintenancePanel);
displayPanel.add("Operations&Trng", operationsPanel);
displayPanel.add("Welcome", welcomePanel);
displayPanel.add("SwitchPanel", switchPanel);
cardLayout1.show(displayPanel, "Welcome");
this.getContentPane().add(displayPanel, new XYConstraints(6, 6,
776, 574));
}

```

```

/**
 * This method is used by the browser to release the ORB
 *
 * @param      none
 * @return     void
 * @exception  default Jbuilder exception
 */
public void stop() {
    try{
        corbaBean.releaseObject();
        System.err.println("Released the orb");
    }
    catch(Exception e){

```



```

        System.err.println("Could not release the orb");
    }
}

/**
 * This method is used by the browser when the user interacts with
 * the applet.
 *
 * @param      e the actionevent generated by the user
 * @return      void
 * @exception   none
 */
public void actionPerformed(ActionEvent e) {

    String command = e.getActionCommand();

    if(command.equals("Administration") && (authorized == true )){
        cardLayout1.show(displayPanel, "Administration");
    }
    else if(command.equals("Budget & Supply") &&
        (authorized == true )){
        cardLayout1.show(displayPanel, "Budget&Supply");
    }
    else if(command.equals("Command") && (authorized == true )){
        cardLayout1.show(displayPanel, "Command");
    }
    else if(command.equals("Maintenance") && (authorized == true )){
        cardLayout1.show(displayPanel, "Maintenance");
    }
    else if(command.equals("Operations & Trng.") &&
        (authorized == true )){
        cardLayout1.show(displayPanel, "Operations&Trng");
    }
    else if(command.equals("Switch Panel") && (authorized == true)){
        cardLayout1.show(displayPanel, "SwitchPanel");
    }
    else if(command.equals("      Welcome      ")){
        setAuthorization(false);
        welcomePanel.nameField.setText("");
        welcomePanel.passwordField.setText("");
        showWelcome();
    }
}

```

```

/**
 * This method is used by the applet to display the AdminPanel
 *
 * @param      none
 * @return     void
 * @exception  none
 */
public void showAdmin() {
    cardLayout1.show(displayPanel, "Administration");
}

/**
 * This method is used by the applet to display the
 * Budget&SupplyPanel
 *
 * @param      none
 * @return     void
 * @exception  none
 */
public void showBudget() {
    cardLayout1.show(displayPanel, "Budget&Supply");
}

/ **
 * This method is used by the applet to display the CommandPanel
 *
 * @param      none
 * @return     void
 * @exception  none
 */
public void showCommand() {
    cardLayout1.show(displayPanel, "Command");
}

/**
 * This method is used by the applet to display the
 * MaintenancePanel
 *
 * @param      none
 * @return     void
 * @exception  none
 */
public void showMaintenance() {
    cardLayout1.show(displayPanel, "Maintenance");
}

```

```

/**
 * This method is used by the applet to display the
 * Operations&TrngPanel
 *
 * @param      none
 * @return     void
 * @exception  none
 */
public void showOperations(){
    cardLayout1.show(displayPanel,"Operations&Trng");
}

/**
 * This method is used by the applet to display the WelcomePanel
 *
 * @param      none
 * @return     void
 * @exception  none
 */
public void showWelcome(){
    cardLayout1.show(displayPanel,"Welcome");
}

/**
 * This method is used by the applet to display the SwitchPanel
 *
 * @param      none
 * @return     void
 * @exception  none
 */
public void showSwitch(){
    cardLayout1.show(displayPanel,"SwitchPanel");
}

}

```

```

//-----
// Filename    : CorbaBean.java
// Authors     : Murat Akbay & Steve Lewis
// Date        : 10/17/1998
// Compiler    : JDK1.1.6 with Symantec JIT Compiler
//-----

import java.awt.*;
import java.applet.*;
import com.sun.java.swing.*;

/**
 * This class is used by the applet to get a reference and invoke
 * methods on the CORBA server
 *
 * @authors Murat Akbay & Steve Lewis
 */
public class CorbaBean {

    Navy.NavSecDispenser    myDispenser;
    Navy.NavSec              myNavSec;

    /**
     * This method is the constructor for the CorbaBean class
     *
     * @param    com.sun.java.swing.JApplet    parent
     * @return    none
     * @exception    default Exception to catch all errors
     */
    public CorbaBean(com.sun.java.swing.JApplet parent) {
        try {
            jbInit(parent);
        }
        catch (Exception ex) {
            ex.printStackTrace();
        }
    }

    /**
     * This method is used get a reference to the naming service and
     * bind to the CORBA server object
     *
     * @param    Applet    parent
     * @return    void
     * @exception    Default Exception to catch all errors.
     */
    private void jbInit(Applet parent) throws Exception {

```

```

try {
    String[] args = null;
    //initialize the orb
    org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(parent , null);

    // Get a reference to the Naming service
    org.omg.CORBA.Object nameServiceObj =
        orb.resolve_initial_references ("NameService");

    if (nameServiceObj == null)
    {
        System.out.println("nameServiceObj = null");
        return;
    }

    org.omg.CosNaming.NamingContext nameService =
        org.omg.CosNaming.NamingContextHelper.narrow
            (nameServiceObj);

    if (nameService == null)
    {
        System.out.println("nameService = null");
        return;
    }

    // bind the NavSec object in the Naming service
    org.omg.CosNaming.NameComponent[] mydispensername =
        {new org.omg.CosNaming.NameComponent("NavSecDispenser", "")};
    myDispenser =
        Navy.NavSecDispenserHelper.narrow
            (nameService.resolve(mydispensername));
    System.out.println("Bind successfull");

    myNavSec= myDispenser.reserveNavSecObject();
    System.out.println("Reserved NavSec Object");

}
catch (Exception e) {
    e.printStackTrace();
}
}

/**
 * This method is used to submit a change of address query to the
 * the database
 *
 * @param      Navy.adressStruct myAddressStruct
 * @return     the struct containing the address info
 * @exception  Default Exception to catch all errors.
 */
public synchronized boolean submitAddressChange
    (Navy.adressStruct myAddressStruct){

```

```

        boolean result;
        try{
            result = myNavSec.addAddress(myAddressStruct);
            System.out.println("Query run successfully");
            return result;

        }
        catch(Exception e){
            System.out.println("Error in processing query");
            return false;
        }
    }

/**
 * This method is used to add a family member to the database
 *
 * @param      Navy.familyStruct  myFamilyStruct
 * @return     the struct containing the family member info
 * @exception  Default Exception to catch all errors.
 */
public synchronized boolean addFamily
                                (Navy.familyStruct myFamilyStruct){
    boolean result;
    try{
        result = myNavSec.addFamily(myFamilyStruct);
        System.out.println("Query run successfully");
        return result;

    }
    catch(Exception e){
        System.out.println("Error in processing query");
        return false;
    }
}

/**
 * This method is used to add a spouse to the database
 *
 * @param      Navy.spouseStruct  mySpouseStruct
 * @return     the struct containing spouse info
 * @exception  Default Exception to catch all errors.
 */
public synchronized boolean addSpouse
                                (Navy.spouseStruct  mySpouseStruct){
    boolean result;
    try{
        result = myNavSec.addSpouse(mySpouseStruct);
        System.out.println("Query run successfully");
        return result;
    }
}

```

```

    }
    catch(Exception e){
        System.out.println("Error in processing query");
        return false;
    }
}

/**
 * This method is used to add PRT info to the database
 *
 * @param      Navy.prtStruct  myPrtStruct
 * @return      the struct containing PRT info
 * @exception   Default Exception to catch all errors.
 */
public synchronized boolean addPrt(Navy.prtStruct  myPrtStruct){
    boolean result;
    try{
        result = myNavSec.addPrt(myPrtStruct);
        System.out.println("Query run successfully");
        return result;
    }
    catch(Exception e){
        System.out.println("Error in processing query");
        return false;
    }
}

/**
 * This method is used to add a members leave data to the database
 *
 * @param      Navy.leaveStruct  myLeaveStruct
 * @return      the struct containing a members leave info
 * @exception   Default Exception to catch all errors.
 */
public synchronized boolean addLeave
                                (Navy.leaveStruct  myLeaveStruct){
    boolean result;
    try{
        result = myNavSec.addLeave(myLeaveStruct);
        System.out.println("Query run successfully");
        return result;
    }
    catch(Exception e){
        System.out.println("Error in processing query");
        return false;
    }
}

```

```

/**
 * This method is used to add a members NEC info to the database
 *
 * @param      Navy.necStruct  myNecStruct
 * @return     the struct containing the members NEC info
 * @exception  Default Exception to catch all errors.
 */
public synchronized boolean addNec(Navy.necStruct myNecStruct){
    boolean result;
    try{
        result = myNavSec.addNec(myNecStruct);
        System.out.println("Query run successfully");
        return result;
    }
    catch(Exception e){
        System.out.println("Error in processing query");
        return false;
    }
}

/**
 * This method is used to add new minor property to the database
 *
 * @param      Navy.propertyStruct  myPropertyStruct
 * @return     the struct containing the minor property info
 * @exception  Default Exception to catch all errors.
 */
public synchronized boolean addProperty
    (Navy.propertyStruct myPropertyStruct){
    boolean result;
    try{
        result = myNavSec.addProperty(myPropertyStruct);
        System.out.println("Query run successfully");
        return result;
    }
    catch(Exception e){
        System.out.println("Error in processing query");
        return false;
    }
}

/**
 * This method is used to add a department to the database
 *
 * @param      Navy.departmentStruct  myDepartmentStruct
 * @return     the struct containing department info
 * @exception  Default Exception to catch all errors.
 */

```



```

public synchronized boolean addDepartment
    (Navy.departmentStruct myDepartmentStruct){
    boolean result;
    try{
        result = myNavSec.addDepartment(myDepartmentStruct);
        System.out.println("Query run successfully");
        return result;
    }
    catch(Exception e){
        System.out.println("Error in processing query");
        return false;
    }
}

/**
 * This method is used to add a division to the database
 *
 * @param      Navy.divisionStruct  myDivisionStruct
 * @return     the struct containing division info
 * @exception  Default Exception to catch all errors.
 */
public synchronized boolean addDivision
    (Navy.divisionStruct myDivisionStruct){
    boolean result;
    try{
        result = myNavSec.addDivision(myDivisionStruct);
        System.out.println("Query run successfully");
        return result;
    }
    catch(Exception e){
        System.out.println("Error in processing query");
        return false;
    }
}

/**
 * This method is used to add maintenance info to the database
 *
 * @param      Navy.maintenanceStruct  myMaintenanceStruct
 * @return     the struct containing maintenance completed info
 * @exception  Default Exception to catch all errors.
 */
public synchronized boolean addMaintLog
    (Navy.maintenanceStruct myMaintenanceStruct){
    boolean result;
    try{
        result = myNavSec.addMaintLog(myMaintenanceStruct);
        System.out.println("Query run successfully");
    }
}

```

```

        return result;
    }
    catch(Exception e){
        System.out.println("Error in processing query");
        return false;
    }
}

/**
 * This method is used to add a new sailor to the database
 *
 * @param      Navy.sailorStruct  mySailorStruct
 * @return     the struct containing the sailors info
 * @exception  Default Exception to catch all errors.
 */
public synchronized boolean addSailor
                                (Navy.sailorStruct mySailorStruct){
    boolean result;
    try{
        result = myNavSec.addSailor(mySailorStruct);
        System.out.println("Query run successfully");
        return result;
    }
    catch(Exception e){
        System.out.println("Error in processing query");
        return false;
    }
}

/**
 * This method is used to add sea mission info to the database
 *
 * @param      Navy.seamissionStruct  mySeamissionStruct
 * @return     the struct containing the mission info
 * @exception  Default Exception to catch all errors.
 */
public synchronized boolean addSeaMission
                                (Navy.seamissionStruct mySeamissionStruct){
    boolean result;
    try{
        result = myNavSec.addSeaMission(mySeamissionStruct);
        System.out.println("Query run successfully");
        return result;
    }
    catch(Exception e){
        System.out.println("Error in processing query");
        return false;
    }
}

```

```

    }

    /**
     * This method is used to add air mission info to the database
     *
     * @param      Navy.airmissionStruct  myAirmissionStruct
     * @return      the struct containing the mission info
     * @exception   Default Exception to catch all errors.
     */
    public synchronized boolean addAirMission
        (Navy.airmissionStruct myAirmissionStruct){
        boolean result;
        try{
            result = myNavSec.addAirMission(myAirmissionStruct);
            System.out.println("Query run successfully");
            return result;

        }
        catch(Exception e){
            System.out.println("Error in processing query");
            return false;
        }
    }

    /**
     * This method is used to change department info in the database
     *
     * @param      Navy.departmentStruct  myDepartmentStruct
     * @return      the struct containing the new department info
     * @exception   Default Exception to catch all errors.
     */
    public synchronized boolean changeDepartment
        (Navy.departmentStruct myDepartmentStruct){
        boolean result;
        try{
            result = myNavSec.changeDeptInfo(myDepartmentStruct);
            System.out.println("Query run successfully");
            return result;

        }
        catch(Exception e){
            System.out.println("Error in processing query");
            return false;
        }
    }
}

```

```

/**
 * This method is used to provide authorization for access to
 * the database
 *
 * @param      String userName
 * @param      String password
 * @return     boolean
 * @exception  none
 */
public synchronized boolean getAuthorization(String userName,
                                             String password){
    boolean result;
    try{
        result = myNavSec.getAuthorization(userName, password);
        System.out.println("Query run successfully");
        return result;
    }
    catch(Exception e){
        System.out.println("Error in processing query");
        return false;
    }
}

/**
 * This method is used to change division info in the database
 *
 * @param      Navy.divisionStruct myDivisionStruct
 * @return     the struct containing new division info
 * @exception  Default Exception to catch all errors.
 */
public synchronized boolean changeDivision
(Navy.divisionStruct myDivisionStruct){
    boolean result;
    try{
        result = myNavSec.changeDivisionInfo(myDivisionStruct);
        System.out.println("Query run successfully");
        return result;
    }
    catch(Exception e){
        System.out.println("Error in processing query");
        return false;
    }
}

```

```

/**
 * This method is used to retrieve family member info from the
 * database
 *
 * @param      String sailorSsn, String lastName
 * @return     information about a specific sailors family members
 * @exception  Default Exception to catch all errors.
 */
public synchronized Navy.familydescStruct[]getFamilyMembers
    (String sailorSsn,String lastName){
    Navy.familydescStruct[] result = null;
    try{
        result = myNavSec.getFamilyMembers(sailorSsn,lastName);
        System.out.println("Query run successfully");
        return result;
    }
    catch(Exception e){
        System.out.println("Error in processing query");
        return null;
    }
}

/**
 * This method is used to retrieve spouse info from the
 * database from a specific sailors data
 *
 * @param      String sailorSsn, String lastName
 * @return     information about a specific sailors spouse
 * @exception  Default Exception to catch all errors.
 */
public synchronized Navy.spousedescStruct getSpouse
    (String sailorSsn,String lastName){
    Navy.spousedescStruct result = null;
    try{
        result = myNavSec.getSpouse(sailorSsn,lastName);
        System.out.println("Query run successfully");
        return result;
    }
    catch(Exception e){
        System.out.println("Error in processing query");
        return null;
    }
}

```

```

/**
 * This method is used to retrieve leave info from the
 * database regarding a specific sailor
 *
 * @param      String sailorSsn,  String lastName
 * @return      information about a specific sailors leave data
 * @exception   Default Exception to catch all errors.
 */
public synchronized Navy.leavedescStruct[] getLeaves
    (String sailorSsn,String lastName){
    Navy.leavedescStruct[] result = null;
    try{
        result = myNavSec.getLeaveData(sailorSsn,lastName);
        System.out.println("Query run successfully");
        return result;

    }
    catch(Exception e){
        System.out.println("Error in processing query");
        return null;
    }
}

```

```

/**
 * This method is used to retrieve PRT info from the
 * database regarding a specific sailor
 *
 * @param      String sailorSsn,  String lastName
 * @return      information about a specific sailors PRT data
 * @exception   Default Exception to catch all errors.
 */
public synchronized Navy.prtdescStruct[] getPrtResults
    (String sailorSsn,String lastName){
    Navy.prtdescStruct[] result = null;
    try{
        result = myNavSec.getPrtResults(sailorSsn,lastName);
        System.out.println("Query run successfully");
        return result;

    }
    catch(Exception e){
        System.out.println("Error in processing query");
        return null;
    }
}

```

```

/**
 * This method is used to retrieve info from the
 * database regarding a specific sailors deployment time
 *
 * @param      String sailorSsn, String lastName, String startDate,
 *              String stopDate
 * @return      information about a specific sailors deployment time
 * @exception   Default Exception to catch all errors.
 */
public synchronized Navy.deploymentStruct[] getDaysDeployed
    (String sailorSsn,String lastName,
     String startDate,String stopDate){
    Navy.deploymentStruct[] result = null;
    try{
        result = myNavSec.getDaysDeployed(sailorSsn,lastName,
                                           startDate,stopDate);
        System.out.println("Query run successfully returning"+ result);
        return result;
    }
    catch(Exception e){
        System.out.println("Error in processing query");
        e.printStackTrace();
        return null;
    }
}

```

```

/**
 * This method is used to retrieve info from the
 * database regarding minor property listings
 *
 * @param      String commandId
 * @return      information about a specific commands property listing
 * @exception   Default Exception to catch all errors.
 */
public synchronized Navy.propertydescStruct[] getPropertyList
    (String commandId){
    Navy.propertydescStruct[] result = null;
    try{
        result = myNavSec.getPropertyList(commandId);
        System.out.println("Query run successfully returning"+ result);
        return result;
    }
    catch(Exception e){
        System.out.println("Error in processing query");
        e.printStackTrace();
        return null;
    }
}

```

```

/**
 * This method is used to retrieve info from the
 * database regarding specific mission information
 *
 * @param      String platform,  String typeOfMission, String
 *              missionArea, String startDate, String stopDate
 * @return      information about a specific mission
 * @exception   Default Exception to catch all errors.
 */
public synchronized Navy.seamissiondescStruct[] getSeaMissions
    (String platform, String typeOfMission,
     String missionArea, String startDate,
     String stopDate){
    Navy.seamissiondescStruct[] result = null;
    try{
        result = myNavSec.getSeaMissions
            (platform,typeOfMission,missionArea,
             startDate,stopDate);
        System.out.println("Query run successfully returning"+ result);
        return result;
    }
    catch(Exception e){
        System.out.println("Error in processing query");
        e.printStackTrace();
        return null;
    }
}

```

```

/**
 * This method is used to retrieve info from the
 * database regarding specific mission information
 *
 * @param      String platform, String startDate, String stopDate
 * @return      information about a specific mission
 * @exception   Default Exception to catch all errors.
 */
public synchronized Navy.seamissiondescStruct[]
    getSeaMissionsByPlatform(String platform,String startDate,
                             String stopDate){
    Navy.seamissiondescStruct[] result = null;
    try{
        result = myNavSec.getSeaMissionsByPlatform(platform,
            startDate,stopDate);
        System.out.println("Query run successfully returning"+ result);
        return result;
    }
    catch(Exception e){
        System.out.println("Error in processing query");
        e.printStackTrace();
    }
}

```



```

        return null;
    }
}

/**
 * This method is used to retrieve info from the
 * database regarding a specific sea mission by area
 *
 * @param      String missionArea, String startDate, String stopDate
 * @return     information about a specific sea mission
 * @exception  Default Exception to catch all errors.
 */
public synchronized Navy.seamissiondescStruct[] getSeaMissionsByArea
        (String missionArea,String startDate,
         String stopDate){
    Navy.seamissiondescStruct[] result = null;
    try{
        result = myNavSec.getSeaMissionsByArea(missionArea,
                                                startDate,stopDate);
        System.out.println("Query run successfully returning"+ result);
        return result;
    }
    catch(Exception e){
        System.out.println("Error in processing query");
        e.printStackTrace();
        return null;
    }
}

/**
 * This method is used to retrieve info from the
 * database regarding a specific sea mission by type
 *
 * @param      String typeOfMission, String startDate, String stopDate
 * @return     information about a specific sea mission
 * @exception  Default Exception to catch all errors.
 */
public synchronized Navy.seamissiondescStruct[] getSeaMissionsByType
        (String typeOfMission, String startDate,
         String stopDate){
    Navy.seamissiondescStruct[] result = null;
    try{
        result = myNavSec.getSeaMissionsByType(typeOfMission,
                                                startDate,stopDate);
        System.out.println("Query run successfully returning"+ result);
        return result;
    }
    catch(Exception e){

```

```

        System.out.println("Error in processing query");
        e.printStackTrace();
        return null;
    }
}

/**
 * This method is used to retrieve info from the
 * database regarding a specific sea mission exercise
 *
 * @param      String exerciseName, String startDate, String stopDate
 * @return     information about a specific sea exercise
 * @exception  Default Exception to catch all errors.
 */
public synchronized Navy.seamissiondescStruct[]
    getSeaMissionsByExercise
    (String exerciseName,String startDate,
     String stopDate){
    Navy.seamissiondescStruct[] result = null;
    try{
        result = myNavSec.getSeaMissionsByExercise(exerciseName,
                                                    startDate,stopDate);
        System.out.println("Query run successfully returning"+ result);
        return result;
    }
    catch(Exception e){
        System.out.println("Error in processing query");
        e.printStackTrace();
        return null;
    }
}

/**
 * This method is used to retrieve info from the
 * database regarding a specific sea mission by platform name
 *
 * @param      String platform, String startDate, String stopDate
 * @return     information about a specific sea mission
 * @exception  Default Exception to catch all errors.
 */
public synchronized Navy.seamissiondescStruct[]
    getSeaMissionsByPlatformName(String platform,String startDate,
                                   String stopDate){
    Navy.seamissiondescStruct[] result = null;
    try{
        result = myNavSec.getSeaMissionsByPlatformName
            (platform,startDate,stopDate);

        System.out.println("Query run successfully ");
    }
}

```

```

        return result;
    }
    catch(Exception e){
        System.out.println("Error in processing query");
        e.printStackTrace();
        return null;
    }
}

/**
 * This method is used to retrieve info from the
 * database regarding a specific air mission
 *
 * @param      String Squadron, String platform, String typeMission,
 *             String missionArea, String startDate, String stopDate
 * @return     information about a specific air mission
 * @exception  Default Exception to catch all errors.
 */
public synchronized Navy.airmissiondescStruct[]
    getAirMissions(String squadron,String platform,
                    String typeMission,
                    String missionArea, String startDate,
                    String stopDate){
    Navy.airmissiondescStruct[] result = null;
    try{
        result = myNavSec.getAirMissions
            (squadron,platform,typeMission,missionArea,
            startDate,stopDate);

        System.out.println("Query run successfully ");
        return result;
    }
    catch(Exception e){
        System.out.println("Error in processing query");
        e.printStackTrace();
        return null;
    }
}

/**
 * This method is used to retrieve info from the
 * database regarding a specific air mission by platform number
 *
 * @param      String platformNo, String startDate, String stopDate
 * @return     information about a specific air mission
 * @exception  Default Exception to catch all errors.
 */
public synchronized Navy.airmissiondescStruct[]

```

```

        getAirMissionByPlatformNo
        (String platformNo, String startDate, String stopDate) {
Navy.airmissiondescStruct[] result = null;
try{
    result = myNavSec.getAirMisByPlatformNo
                    (platformNo, startDate, stopDate);

    System.out.println("Query run successfully ");
    return result;

}
catch(Exception e){
    System.out.println("Error in processing query");
    e.printStackTrace();
    return null;
}
}

/**
 * This method is used to retrieve info from the
 * database regarding a specific air mission by area
 *
 * @param      String missionArea, String startDate, String stopDate
 * @return     information about a specific air mission
 * @exception  Default Exception to catch all errors.
 */
public synchronized Navy.airmissiondescStruct[]
    getAirMissionByArea(String missionArea, String startDate,
                        String stopDate){
Navy.airmissiondescStruct[] result = null;
try{
    result = myNavSec.getAirMisByArea
                    (missionArea, startDate, stopDate);

    System.out.println("Query run successfully ");
    return result;

}
catch(Exception e){
    System.out.println("Error in processing query");
    e.printStackTrace();
    return null;
}
}

```

```

/**
 * This method is used to retrieve info from the
 * database regarding a specific air mission by type
 *
 * @param      String typeMission, String startDate, String stopDate
 * @return      information about a specific air mission
 * @exception   Default Exception to catch all errors.
 */
public synchronized Navy.airmissiondescStruct[]
    getAirMissionByType(String typeMission, String startDate,
                        String stopDate){
    Navy.airmissiondescStruct[] result = null;
    try{
        result = myNavSec.getAirMisByType
            (typeMission,startDate,stopDate);

        System.out.println("Query run successfully ");
        return result;

    }
    catch(Exception e){
        System.out.println("Error in processing query");
        e.printStackTrace();
        return null;
    }
}

/**
 * This method is used to retrieve info from the
 * database regarding a specific air mission by exercise
 *
 * @param      String exerciseName, String startDate, String stopDate
 * @return      information about a specific air mission
 * @exception   Default Exception to catch all errors.
 */
public synchronized Navy.airmissiondescStruct[]
    getAirMissionByExercise(String exerciseName,String startDate,
                            String stopDate){
    Navy.airmissiondescStruct[] result = null;
    try{
        result = myNavSec.getAirMisByExercise
            (exerciseName,startDate,stopDate);

        System.out.println("Query run successfully ");
        return result;

    }
    catch(Exception e){
        System.out.println("Error in processing query");
        e.printStackTrace();
        return null;
    }
}

```

```

    }
}

/**
 * This method is used to retrieve info from the
 * database regarding a specific air mission by squadron
 *
 * @param      String squadron, String startDate, String stopDate
 * @return     information about a specific air mission
 * @exception  Default Exception to catch all errors.
 */
public synchronized Navy.airmissiondescStruct[]
    getAirMissionBySquadron(String squadron,String startDate,
                           String stopDate){
    Navy.airmissiondescStruct[] result = null;
    try{
        result = myNavSec.getAirMisBySquadron
            (squadron,startDate,stopDate);

        System.out.println("Query run successfully ");
        return result;

    }
    catch(Exception e){
        System.out.println("Error in processing query");
        e.printStackTrace();
        return null;
    }
}

/**
 * This method is used to retrieve info from the
 * database regarding the demographics of a specific command by gender
 *
 * @param      String commandId, String sex
 * @return     information about a specific commands demographics
 * @exception  Default Exception to catch all errors.
 */
public synchronized Navy.demographicStruct[]
    getComDemByGender(String commandId, String sex){
    Navy.demographicStruct[] result = null;
    try{
        result = myNavSec.getComDemByGender(commandId,sex);

        System.out.println("Query run successfully ");
        return result;

    }
    catch(Exception e){
        System.out.println("Error in processing query");

```

```

        e.printStackTrace();
        return null;
    }
}

/**
 * This method is used to retrieve info from the
 * database regarding a specific commands demographics by nec
 *
 * @param      String commandId, String nec1, String nec2,
 *             String nec3
 * @return     information about a specific commands demographics
 * @exception  Default Exception to catch all errors.
 */
public synchronized Navy.demographicStruct[]
    getComDemByNec(String commandId, String nec1, String nec2,
        String nec3){
    Navy.demographicStruct[] result = null;
    try{
        result = myNavSec.getComDemByNec(commandId,nec1,nec2,nec3);

        System.out.println("Query run successfully ");
        return result;

    }
    catch(Exception e){
        System.out.println("Error in processing query");
        e.printStackTrace();
        return null;
    }
}

/**
 * This method is used to retrieve info from the
 * database regarding a specific commands demographics by rank
 *
 * @param      String sailorSsn, String paygrade
 * @return     information about a specific commands demographics
 * @exception  Default Exception to catch all errors.
 */
public synchronized Navy.demographicStruct[]
    getComDemByRank(String sailorSsn, String paygrade){
    Navy.demographicStruct[] result = null;
    try{
        result = myNavSec.getComDemByRank(sailorSsn,paygrade);

        System.out.println("Query run successfully ");
        return result;

    }
}

```

```

        catch(Exception e){
            System.out.println("Error in processing query");
            e.printStackTrace();
            return null;
        }
    }

/**
 * This method is used to retrieve info from the
 * database regarding a specific commands emergency list
 *
 * @param      String commandId
 * @return     information about a specific commands roster
 * @exception  Default Exception to catch all errors.
 */
public synchronized Navy.emergencyStruct[]
    getEmergencyList(String commandId){
    Navy.emergencyStruct[] result = null;
    try{
        result = myNavSec.getEmergencyList(commandId);

        System.out.println("Query run successfully ");
        return result;

    }
    catch(Exception e){
        System.out.println("Error in processing query");
        e.printStackTrace();
        return null;
    }
}

/**
 * This method is used to retrieve info from the
 * database regarding a specific commands maintenance data
 *
 * @param      String commandId, String departmentId, String
 *              startDate, String stopDate
 * @return     information about a specific commands maintenance
 * @exception  Default Exception to catch all errors.
 */
public synchronized Navy.commaintenanceStruct[]
    getMaintByCommand(String commandId,
                      String departmentId,
                      String startDate, String stopDate){
    Navy.commaintenanceStruct[] result = null;
    try{
        result = myNavSec.getMaintByCommand(commandId, departmentId,
                                             startDate, stopDate);
    }
}

```



```

        System.out.println("Query run successfully ");
        return result;
    }
    catch(Exception e){
        System.out.println("Error in processing query");
        e.printStackTrace();
        return null;
    }
}

/**
 * This method is used to retrieve info from the
 * database regarding a specific commands maintenance by item
 *
 * @param      String item
 * @return     maint information about a specific item in a command
 * @exception  Default Exception to catch all errors.
 */
public synchronized Navy.itemmaintenanceStruct[]
    getMaintByItem(String item){
    Navy.itemmaintenanceStruct[] result = null;
    try{
        result = myNavSec.getMaintByItem(item);

        System.out.println("Query run successfully ");
        return result;
    }
    catch(Exception e){
        System.out.println("Error in processing query");
        e.printStackTrace();
        return null;
    }
}

/**
 * This method is used to release the CORBA server object
 *
 * @param      none
 * @return     void
 * @exception  Default Exception to catch all errors.
 */
public void releaseObject( ){
    try{
        myDispenser.releaseNavSecObject(myNavSec);
    }
    catch(Exception e){
    }
}
}

```

```
//-----
// Filename   : AdminPanel.java
// Authors    : Murat Akbay & Steve Lewis
// Date       : 10/17/1998
// Compiler   : JDK1.1.6 with Symantec JIT Compiler
//-----
```

```
import java.awt.*;
import java.applet.*;
import borland.jbcl.layout.*;
import borland.jbcl.control.*;
import com.sun.java.swing.*;
import java.awt.event.*;
```

```
/**
 * This class is used to create the administration GUI panel
 *
 * @authors Murat Akbay & Steve Lewis
 */
```

```
public class AdminPanel extends JPanel{

    XYLayout xYLayout1 = new XYLayout();
    JTabbedPane jTabbedPane1 = new JTabbedPane();
    ChangeAddressBean jPanel1;
    AddFamilyBean jPanel2;
    AddSpouseBean jPanel3;
    AddPrtBean jPanel4;
    AddLeaveBean jPanel5;
    AddSailorBean jPanel6;
    AddNecBean jPanel7;
    GetFamilyMemberBean jPanel8;
    GetSpouseBean jPanel9;
    GetLeaveBean jPanel10;
    GetPrtBean jPanel11;
    GetDaysDeployedBean jPanel12;
    BevelPanel headerPanel = new BevelPanel();
    XYLayout xYLayout2 = new XYLayout();
    JLabel jLabel1 = new JLabel();
    private Gui parentApplet;
```

```
/**
 * This is the constructor used to create all of the panels in the
 * admin panel GUI interface
 *
 * @param      Gui parent
 * @return     void
 * @exception  Default Exception to catch all errors.
 */
```

```
public AdminPanel(Gui parent) {
```

```

parentApplet = parent;
//create the panels
jPanel1 = new ChangeAddressBean(parentApplet);
jPanel2 = new AddFamilyBean(parentApplet);
jPanel3 = new AddSpouseBean(parentApplet);
jPanel4 = new AddPrtBean(parentApplet);
jPanel5= new AddLeaveBean(parentApplet);
jPanel6 = new AddSailorBean(parentApplet);
jPanel7 = new AddNecBean(parentApplet);
jPanel8 = new GetFamilyMemberBean(parentApplet);
jPanel9 = new GetSpouseBean(parentApplet);
jPanel10 = new GetLeaveBean(parentApplet);
jPanel11 = new GetPrtBean(parentApplet);
jPanel12 = new GetDaysDeployedBean(parentApplet);
//initialize the panel
try {
    jbInit();
}
catch (Exception ex) {
    ex.printStackTrace();
}
}

```

```

/**
 * This method is used to initialize the admin panels
 *
 * @param      none
 * @return     void
 * @exception  Default Exception to catch all errors.
 */
private void jbInit() throws Exception {
    headerPanel.setLayout(xYLayout2);
    jLabel1.setText("NAVY SECURITY GROUP " + "EIS ADMINISTRATION");
    this.setLayout(xYLayout1);
    xYLayout1.setHeight(576);
    this.setBackground(Color.lightGray);
    xYLayout1.setWidth(776);
    //add the panels
    this.add(jTabbedPanel, new XYConstraints(11, 41, 745, 521));
    this.add(headerPanel, new XYConstraints(11, 4, 745, 29));
    headerPanel.add(jLabel1, new XYConstraints(91, 3, 568, 23));
    JScrollPane scrollpane = new JScrollPane(jPanel6,
        JScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS,
        JScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER);
    scrollpane.setPreferredSize(new Dimension(656,460));
    jTabbedPanel.addTab("Change Sailor Address", jPanel1);
    jTabbedPanel.addTab("Add Family Member ", jPanel2);
    jTabbedPanel.addTab("Add Spouse " , jPanel3);
    jTabbedPanel.addTab("Add Prt Data", jPanel4);
    jTabbedPanel.addTab("Add Leave Data", jPanel5);
    jTabbedPanel.addTab("Add Sailor ",scrollpane);
}

```

```
jTabbedPanel1.addTab("Update  Nec",jPanel7);  
jTabbedPanel1.addTab("Get  Family  Members",jPanel8);  
jTabbedPanel1.addTab("Get Spouse Of The Sailor",jPanel9);  
jTabbedPanel1.addTab("Get  Leave  Data",jPanel10);  
jTabbedPanel1.addTab("Get  Prt  Results",jPanel11);  
jTabbedPanel1.addTab("Get  Days  Deployed",jPanel12);
```

```
}
```

```
}
```

```

//-----
// Filename      : CommandPanel.java
// Authors       : Murat Akbay & Steve Lewis
// Date          : 10/17/1998
// Compiler      : JDK1.1.6 with Symantec JIT Compiler
//-----

import java.awt.*;
import java.applet.*;
import borland.jbcl.layout.*;
import borland.jbcl.control.*;
import com.sun.java.swing.*;
import java.awt.event.*;

/**
 * This class is used to create the command GUI panel
 *
 * @authors Murat Akbay & Steve Lewis
 */
public class CommandPanel extends JPanel{

    XYLayout xYLayout1 = new XYLayout();
    JTabbedPane jTabbedPane1 = new JTabbedPane();
    AddDepartmentBean jPanel1 ;
    AddDivisionBean jPanel2 ;
    ChangeDepartmentBean jPanel3 ;
    ChangeDivisionBean jPanel4 ;
    GetComDemByGenderBean jPanel5;
    GetComDemByNecBean jPanel6;
    GetComDemByRankBean jPanel7;
    GetEmergencyListBean jPanel8;
    private Gui parentApplet;
    JLabel jLabel1 = new JLabel();
    XYLayout xYLayout2 = new XYLayout();
    BevelPanel headerPanel = new BevelPanel();

    /**
     * This is the constructor used to create all of the panels in the
     * command panel GUI interface
     *
     * @param      Gui parent
     * @return     void
     * @exception   Default Exception to catch all errors.
     */
    public CommandPanel(Gui parent) {
        parentApplet = parent;
        jPanel1 = new AddDepartmentBean(parentApplet);
        jPanel2 = new AddDivisionBean(parentApplet);
        jPanel3 = new ChangeDepartmentBean(parentApplet);
        jPanel4 = new ChangeDivisionBean(parentApplet);
    }
}

```

```

jPanel5 = new GetComDemByGenderBean(parentApplet);
jPanel6 = new GetComDemByNecBean(parentApplet);
jPanel7 = new GetComDemByRankBean(parentApplet);
jPanel8 = new GetEmergencyListBean(parentApplet);
try {
    jbInit();
}
catch (Exception ex) {
    ex.printStackTrace();
}
}

/**
 * This method is used to initialize the command panels
 *
 * @param      none
 * @return     void
 * @exception  Default Exception to catch all errors.
 */
private void jbInit() throws Exception {

    this.setLayout(xYLayout1);
    xYLayout1.setHeight(576);
    jLabel1.setText(" NAVY SECURITY GROUP " + " EIS  COMMAND");
    headerPanel.setLayout(xYLayout2);
    this.setBackground(Color.lightGray);
    xYLayout1.setWidth(776);

    this.add(jTabbedPanel1, new XYConstraints(11, 41, 745, 521));
    this.add(headerPanel, new XYConstraints(11, 4, 745, 29));
    headerPanel.add(jLabel1, new XYConstraints(122, 6, -1, -1));
    jTabbedPanel1.addTab("Add Department", jPanel1);
    jTabbedPanel1.addTab("Add Division ", jPanel2);
    jTabbedPanel1.addTab("Change Department Data" , jPanel3);
    jTabbedPanel1.addTab("Change Division Data " , jPanel4);
    jTabbedPanel1.addTab("View Command Demographics By Gender",
                        jPanel5);
    jTabbedPanel1.addTab("View Command Demographics By Nec" , jPanel6);
    jTabbedPanel1.addTab("View Command Demographics By Rank" , jPanel7);
    jTabbedPanel1.addTab("View Command Emergency List" , jPanel8);

}
}

```

```

//-----
// Filename    : OperationsPanel.java
// Authors     : Murat Akbay & Steve Lewis
// Date        : 10/17/1998
// Compiler    : JDK1.1.6 with Symantec JIT Compiler
//-----

import java.awt.*;
import java.applet.*;
import borland.jbcl.layout.*;
import borland.jbcl.control.*;
import com.sun.java.swing.*;
import java.awt.event.*;

/**
 * This class is used to create the Operations GUI panel
 *
 * @authors Murat Akbay & Steve Lewis
 */
public class OperationsPanel extends JPanel{

    XYLayout xYLayout1 = new XYLayout();
    JTabbedPane jTabbedPane1 = new JTabbedPane();

    AddSeaMissionBean jPanel1 ;
    GetSeaMissionsBean jPanel2;
    GetSeaMissionsByPlatformBean jPanel3;
    GetSeaMissionsByAreaBean jPanel4;
    GetSeaMissionsByTypeBean jPanel5;
    GetSeaMissionsByExerciseBean jPanel6;
    GetSeaMissionsByPlatformNameBean jPanel7;
    AddAirMissionBean jPanel8 ;
    GetAirMissionsBean jPanel9;
    GetAirMissionsByPlatformBean jPanel10;
    GetAirMissionsByAreaBean jPanel11;
    GetAirMissionsByTypeBean jPanel12;
    GetAirMissionsByExerciseBean jPanel13;
    GetAirMissionsBySquadronBean jPanel14;

    private Gui parentApplet;
    JLabel jLabel1 = new JLabel();
    XYLayout xYLayout2 = new XYLayout();
    BevelPanel headerPanel = new BevelPanel();

```

```

/**
 * This is the constructor used to create all of the panels in the
 * Operations panel GUI interface
 *
 * @param      Gui parent
 * @return      void
 * @exception   Default Exception to catch all errors.
 */
public OperationsPanel(Gui parent) {
    parentApplet = parent;
    jPanel1 = new AddSeaMissionBean(parentApplet);
    jPanel2 = new GetSeaMissionsBean(parentApplet);
    jPanel3 = new GetSeaMissionsByPlatformBean(parentApplet);
    jPanel4 = new GetSeaMissionsByAreaBean(parentApplet);
    jPanel5 = new GetSeaMissionsByTypeBean(parentApplet);
    jPanel6 = new GetSeaMissionsByExerciseBean(parentApplet);
    jPanel7 = new GetSeaMissionsByPlatformNameBean(parentApplet);
    jPanel8 = new AddAirMissionBean(parentApplet);
    jPanel9 = new GetAirMissionsBean(parentApplet);
    jPanel10 = new GetAirMissionsByPlatformBean(parentApplet);
    jPanel11 = new GetAirMissionsByAreaBean(parentApplet);
    jPanel12 = new GetAirMissionsByTypeBean(parentApplet);
    jPanel13 = new GetAirMissionsByExerciseBean(parentApplet);
    jPanel14 = new GetAirMissionsBySquadronBean(parentApplet);
    try {
        jbInit();
    }
    catch (Exception ex) {
        ex.printStackTrace();
    }
}

/**
 * This method is used to initialize the operations panels
 *
 * @param      none
 * @return      void
 * @exception   Default Exception to catch all errors.
 */
private void jbInit() throws Exception {
    this.setLayout(xYLayout1);
    xYLayout1.setHeight(576);
    jLabel1.setText("NAVY SECURITY GROUP" + " EIS OPERATIONS &
                                                             TRAINING");
    headerPanel.setLayout(xYLayout2);
    this.setBackground(Color.lightGray);
    xYLayout1.setWidth(776);

    //add the panels
    this.add(jTabbedPane1, new XYConstraints(13, 37, 745, 521));
    this.add(headerPanel, new XYConstraints(11, 4, 745, 29));
}

```



```

headerPanel.add(jLabel1, new XYConstraints(90, 7, -1, -1));
JScrollPane scrollpane = new JScrollPane(jPanel8,
    ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS,
    ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER);
scrollpane.setPreferredSize(new Dimension(656,460));
//Icon orion = new ImageIcon("orion.gif");
jTabbedPane1.addTab(" Add Sea Mission", jPanel1);
jTabbedPane1.addTab(" View Sea Missions", jPanel2);
jTabbedPane1.addTab(" By Platform", jPanel3);
jTabbedPane1.addTab(" By Area", jPanel4);
jTabbedPane1.addTab(" By Type", jPanel5);
jTabbedPane1.addTab(" By Exercise", jPanel6);
jTabbedPane1.addTab(" By Platform Name", jPanel7);
jTabbedPane1.addTab(" Add Air Mission", scrollpane);
jTabbedPane1.addTab(" View Air Missions", jPanel9);
jTabbedPane1.addTab(" By Platform No",jPanel10);
jTabbedPane1.addTab(" By Mission Area",jPanel11);
jTabbedPane1.addTab(" By Type",jPanel12);
jTabbedPane1.addTab(" By Exercise",jPanel13);
jTabbedPane1.addTab(" By Squadron",jPanel14);
}
}

```

```

//-----
// Filename    : MaintenancePanel.java
// Authors     : Murat Akbay & Steve Lewis
// Date        : 10/17/1998
// Compiler    : JDK1.1.6 with Symantec JIT Compiler
//-----

import java.awt.*;
import java.applet.*;
import borland.jbcl.layout.*;
import borland.jbcl.control.*;
import com.sun.java.swing.*;
import java.awt.event.*;

/**
 * This class is used to create the Maintenance GUI panel
 *
 * @authors Murat Akbay & Steve Lewis
 */
public class MaintenancePanel extends JPanel{

    XYLayout xYLayout1 = new XYLayout();
    JTabbedPane jTabbedPane1 = new JTabbedPane();
    AddMaintBean jPanel1 ;
    GetMaintByCommandBean jPanel2;
    GetMaintByItemBean jPanel3;
    private Gui parentApplet;
    JLabel jLabel1 = new JLabel();
    XYLayout xYLayout2 = new XYLayout();
    BevelPanel headerPanel = new BevelPanel();

    /**
     * This is the constructor used to create all of the panels in the
     * Maintenance panel GUI interface
     *
     * @param      Gui parent
     * @return      void
     * @exception   Default Exception to catch all errors.
     */
    public MaintenancePanel(Gui parent) {
        parentApplet = parent;
        jPanel1 = new AddMaintBean(parentApplet);
        jPanel2 = new GetMaintByCommandBean(parentApplet);
        jPanel3 = new GetMaintByItemBean(parentApplet);
        try {
            jbInit();
        }
        catch (Exception ex) {
            ex.printStackTrace();
        }
    }

```

```

    }

    /**
     * This method is used to initialize the maintenance panels
     *
     * @param      none
     * @return     void
     * @exception  Default Exception to catch all errors.
     */
    private void jbInit() throws Exception {

        this.setLayout(xYLayout1);
        xYLayout1.setHeight(576);
        jLabel1.setText
            (" NAVY SECURITY GROUP " + " EIS MAINTENANCE");
        headerPanel.setLayout(xYLayout2);
        this.setBackground(Color.lightGray);
        xYLayout1.setWidth(776);

        //add the panels
        this.add(jTabbedPanel1, new XYConstraints(11, 41, 745, 521));
        this.add(headerPanel, new XYConstraints(11, 4, 745, 29));
        headerPanel.add(jLabel1, new XYConstraints(128, 6, -1, -1));
        JScrollPane scrollpane = new JScrollPane(jPanel1,
            ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS ,
            ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER);
        scrollpane.setPreferredSize(new Dimension(656,460));
        jTabbedPanel1.addTab("Add Maintenance", scrollpane);
        jTabbedPanel1.addTab("View Maintenance By Command", jPanel2);
        jTabbedPanel1.addTab("View Maintenance By Item", jPanel3);
    }
}

```

```

//-----
// Filename   : BudgetPanel.java
// Authors    : Murat Akbay & Steve Lewis
// Date       : 10/17/1998
// Compiler   : JDK1.1.6 with Symantec JIT Compiler
//-----

import java.awt.*;
import java.applet.*;
import borland.jbcl.layout.*;
import borland.jbcl.control.*;
import com.sun.java.swing.*;
import java.awt.event.*;

/**
 * This class is used to create the Budget GUI panel
 *
 * @authors Murat Akbay & Steve Lewis
 */
public class BudgetPanel extends JPanel{

    XYLayout xYLayout1 = new XYLayout();
    JTabbedPane jTabbedPane1 = new JTabbedPane();
    AddPropertyBean jPanel1 ;
    GetPropertyListBean jPanel2 ;
    private Gui parentApplet;
    JLabel jLabel1 = new JLabel();
    XYLayout xYLayout2 = new XYLayout();
    BevelPanel headerPanel = new BevelPanel();

    /**
     * This is the constructor used to create all of the panels in the
     * Budget panel GUI interface
     *
     * @param      Gui parent
     * @return     void
     * @exception  Default Exception to catch all errors.
     */
    public BudgetPanel(Gui parent) {
        parentApplet = parent;
        jPanel1 = new AddPropertyBean(parentApplet);
        jPanel2 = new GetPropertyListBean(parentApplet);
        try {
            jbInit();
        }
        catch (Exception ex) {
            ex.printStackTrace();
        }
    }
}

```

```

/**
 * This method is used to initialize the budget panels
 *
 * @param      none
 * @return     void
 * @exception  Default Exception to catch all errors.
 */
private void jbInit() throws Exception {
    this.setLayout(xYLayout1);
    xYLayout1.setHeight(576);
    jLabel1.setText("NAVY SECURITY GROUP " + "  EIS BUDGET & SUPPLY");
    headerPanel.setLayout(xYLayout2);
    this.setBackground(Color.lightGray);
    xYLayout1.setWidth(776);

    this.add(jTabbedPanel, new XYConstraints(11, 41, 745, 521));
    this.add(headerPanel, new XYConstraints(11, 4, 745, 29));
    headerPanel.add(jLabel1, new XYConstraints(105, 7, -1, -1));
    jTabbedPanel.addTab("Add  Property", jPanel1);
    jTabbedPanel.addTab("View Properties", jPanel2);
}
}

```

```

//-----
// Filename   : SwitchPanel.java
// Authors    : Murat Akbay & Steve Lewis
// Date       : 10/17/1998
// Compiler   : JDK1.1.6 with Symantec JIT Compiler
//-----

import java.awt.*;
import java.awt.event.*;
import com.sun.java.swing.*;
import borland.jbcl.layout.*;
import borland.jbcl.control.*;

/**
 * This class is used to create the Switch GUI panel
 *
 * @authors Murat Akbay & Steve Lewis
 */
public class SwitchPanel extends JPanel implements ActionListener{
    XYLayout xYLayout1 = new XYLayout();
    JButton adminButton = new JButton();
    JButton commandButton = new JButton();
    JButton operationButton = new JButton();
    JButton budgetButton = new JButton();
    JButton maintenanceButton = new JButton();
    JButton welcomeButton = new JButton();
    private Gui parentApplet;
    BevelPanel bevelPanel1 = new BevelPanel();
    JLabel jLabel1 = new JLabel();

    /**
     * This is the constructor used to create all of the panels in the
     * switch panel GUI interface
     *
     * @param      Gui parent
     * @return     void
     * @exception   Default Exception to catch all errors.
     */
    public SwitchPanel(Gui parent) {
        parentApplet = parent;
        try {
            jbInit();
        }
        catch (Exception ex) {
            ex.printStackTrace();
        }
    }
}

```

```

/**
 * This method is used to initialize the switch panel
 *
 * @param      none
 * @return     void
 * @exception  Default Exception to catch all errors.
 */
void jbInit() throws Exception {
    xYLayout1.setHeight(576);
    xYLayout1.setWidth(776);
    adminButton.setText("ADMINISTRATION");
    adminButton.setBackground(new Color(192, 192, 224));
    adminButton.setOpaque(true);
    adminButton.addActionListener(this);
    commandButton.setText("COMMAND");
    commandButton.setBackground(new Color(192, 192, 224));
    commandButton.setOpaque(true);
    commandButton.addActionListener(this);
    operationButton.setText("jButton1");
    operationButton.setLabel("OPERATIONS");
    operationButton.setBackground(new Color(192, 192, 224));
    operationButton.setOpaque(true);
    operationButton.addActionListener(this);
    budgetButton.setText("BUDGET & SUPPLY");
    budgetButton.setBackground(new Color(192, 192, 224));
    budgetButton.setOpaque(true);
    budgetButton.addActionListener(this);
    maintenanceButton.setText("jButton1");
    maintenanceButton.setLabel("MAINTENANCE");
    maintenanceButton.setBackground(new Color(192, 192, 224));
    maintenanceButton.setOpaque(true);
    maintenanceButton.addActionListener(this);
    welcomeButton.setText("jButton1");
    welcomeButton.setLabel("WELCOME");
    welcomeButton.setBackground(new Color(192, 192, 224));
    welcomeButton.setOpaque(true);
    jLabel1.setText(" NAVY SECURITY GROUP " + "    EIS");
    welcomeButton.addActionListener(this);
    this.setLayout(xYLayout1);
    this.add(budgetButton, new XYConstraints(460, 136, 158, 66));
    this.add(maintenanceButton, new XYConstraints(460, 226, 158, 66));
    this.add(welcomeButton, new XYConstraints(461, 315, 158, 66));
    this.add(adminButton, new XYConstraints(152, 137, 158, 66));
    this.add(operationButton, new XYConstraints(154, 316, 158, 66));
    this.add(commandButton, new XYConstraints(153, 227, 158, 66));
    this.add(bevelPanel1, new XYConstraints(72, 44, 632, 57));
    bevelPanel1.add(jLabel1, new XYConstraints(81, 14, 496, 27));
}

```

```

/**
 * This method is used perform the actions required when the user
 * selects one of the switches on the panel.
 *
 * @param      e  action performed by the user
 * @return      void
 * @exception   Default Exception to catch all errors.
 */
public void actionPerformed(ActionEvent e) {

    String command = e.getActionCommand();

    if(command.equals("ADMINISTRATION")){
        parentApplet.showAdmin();
    }
    else if(command.equals("BUDGET & SUPPLY")){
        parentApplet.showBudget();
    }
    else if(command.equals("COMMAND")){
        parentApplet.showCommand();
    }
    else if(command.equals("MAINTENANCE")){
        parentApplet.showMaintenance();
    }
    else if(command.equals("OPERATIONS")){
        parentApplet.showOperations();
    }
    else if(command.equals("WELCOME")){
        parentApplet.setAuthorization(false);
        parentApplet.welcomePanel.nameField.setText("");
        parentApplet.welcomePanel.passwordField.setText("");
        parentApplet.showWelcome();
    }
}
}

```



```

//-----
// Filename      : WelcomePanel.java
// Authors       : Murat Akbay & Steve Lewis
// Date          : 10/17/1998
// Compiler       : JDK1.1.6 with Symantec JIT Compiler
//-----

import java.awt.*;
import java.applet.*;
import borland.jbcl.layout.*;
import borland.jbcl.control.*;
import com.sun.java.swing.*;
import java.awt.event.*;

public class WelcomePanel extends JPanel{

    XYLayout xYLayout1 = new XYLayout();
    JPanel entryPanel = new JPanel();
    private Gui parentApplet;
    XYLayout xYLayout2 = new XYLayout();
    BevelPanel bevelPanel1 = new BevelPanel();
    JLabel jLabel1 = new JLabel();
    JPanel jPanel2 = new JPanel();
    XYLayout xYLayout3 = new XYLayout();
    JLabel jLabel2 = new JLabel();
    JLabel jLabel3 = new JLabel();
    JPasswordField nameField = new JPasswordField();
    JPasswordField passwordField = new JPasswordField();
    JButton submitButton = new JButton();

    public WelcomePanel(Gui parent) {
        parentApplet = parent;
        try {
            jbInit();
        }
        catch (Exception ex) {
            ex.printStackTrace();
        }
    }

    public boolean getAuthorization(){
        boolean returnValue = false;

        if(parentApplet.corbaBean.getAuthorization(nameField.getText().trim(),
                                                    passwordField.getText().trim())){
            returnValue = true;
        }
        return returnValue;
    }

    private void jbInit() throws Exception {

```

```

this.setLayout(xYLayout1);
xYLayout1.setHeight(576);
jLabel1.setText("      WELCOME    TO    NAVY    SECURITY    GROUP " +
" ENTERPRISE    INFORMATION    SYSTEM");
jLabel2.setText("LOG ON NAME");
jLabel3.setText("PASSWORD");
nameField.setText("jPasswordField1");
passwordField.setText("jPasswordField2");
submitButton.setText("SUBMIT");
submitButton.setLabel(" ENTER");
submitButton.setBackground(new Color(192, 192, 209));
submitButton.setOpaque(true);
submitButton.addActionListener(new java.awt.event.ActionListener()
{
    public void actionPerformed(ActionEvent e) {
        submitButton_actionPerformed(e);
    }
});
//logoLabel.setOpaque(true);
jPanel2.setLayout(xYLayout3);
entryPanel.setLayout(xYLayout2);
this.setBackground(Color.lightGray);
xYLayout1.setWidth(776);

//add the panels
this.add(entryPanel, new XYConstraints(22, 69, 722, 439));
entryPanel.add(bevelPanel1, new XYConstraints(11, 272, 694, 63));
bevelPanel1.add(jLabel1, new XYConstraints(27, 19, 644, 26));
entryPanel.add(jPanel2, new XYConstraints(165, 352, 348, 73));
jPanel2.add(jLabel2, new XYConstraints(5, 9, 114, 22));
jPanel2.add(jLabel3, new XYConstraints(5, 40, 114, 22));
jPanel2.add(nameField, new XYConstraints(178, 12, 160, 22));
jPanel2.add(passwordField, new XYConstraints(178, 41, 160, 22));
this.add(submitButton, new XYConstraints(354, 528, -1, -1));
//entryPanel.add(logoLabel, new XYConstraints(165, 6, 369, 254));

}

void submitButton_actionPerformed(ActionEvent e) {
    if(getAuthorization() == true){
        this.parentApplet.setAuthorization(true);
        this.parentApplet.showSwitch();
    }
}
}

```

```

//-----
// Filename      : AddFamilyBean.java
// Authors       : Murat Akbay & Steve Lewis
// Date          : 10/17/1998
// Compiler      : JDK1.1.6 with Symantec JIT Compiler
//-----

import java.awt.*;
import borland.jbcl.layout.*;
import com.sun.java.swing.*;
import java.awt.event.*;
import java.util.*;
import java.io.*;

/**
 * This class is used to create the AddFamilyBean GUI panel
 *
 * @authors Murat Akbay & Steve Lewis
 */
public class AddFamilyBean extends JPanel implements ActionListener{
    XYLayout xYLayout1 = new XYLayout();
    JPanel dataPanel = new JPanel();
    JPanel labelPanel = new JPanel();
    JLabel ssnLabel = new JLabel();
    JPanel headerPanel = new JPanel();
    JLabel addPrtLabel = new JLabel();
    GridBagLayout gridBagLayout3 = new GridBagLayout();
    XYLayout xYLayout2 = new XYLayout();
    JPanel buttonPanel = new JPanel();
    JPanel blankPanel = new JPanel();
    JButton submitButton = new JButton();
    JButton clearButton = new JButton();
    XYLayout xYLayout3 = new XYLayout();
    JLabel nameLabel = new JLabel();
    JLabel birthLabel = new JLabel();
    JLabel sexLabel = new JLabel();
    JLabel sailorSsnLabel = new JLabel();
    JTextField jTextField1 = new JTextField();
    JTextField ssnField = new JTextField();
    JTextField nameField = new JTextField();
    JTextField birthField = new JTextField();
    JTextField sexField = new JTextField();
    JTextField sailorSsnField = new JTextField();
    JButton cancelButton = new JButton();
    GridBagLayout gridBagLayout1 = new GridBagLayout();
    private Gui parentApplet;

```

```

/**
 * This is the constructor used to create the
 * AddFamilyBean panel GUI interface
 *
 * @param      Gui parent
 * @return      void
 * @exception   Default Exception to catch all errors.
 */
public AddFamilyBean(Gui parent) {
    parentApplet = parent;
    try {
        jbInit();
    }
    catch (Exception ex) {
        ex.printStackTrace();
    }
}

/**
 * This method is used to initialize the AddFamilyBean panel
 *
 * @param      none
 * @return      void
 * @exception   Default Exception to catch all errors.
 */
private void jbInit() throws Exception {
    addPrtLabel.setText("      ADD FAMILY MEMBER FORM");
    submitButton.setText("SUBMIT");
    clearButton.setText("CLEAR");
    nameLabel.setText(" NAME");
    nameLabel.setOpaque(true);
    birthLabel.setText(" BIRTH DATE");
    birthLabel.setOpaque(true);
    sexLabel.setText(" SEX");
    sexLabel.setOpaque(true);
    sailorSsnLabel.setText(" SAILOR SSN");
    sailorSsnLabel.setOpaque(true);
    jTextField1.setText(" ");
    cancelButton.setText("CANCEL");
    ssnField.setText(" ");
    nameField.setText(" ");
    birthField.setText(" ");
    sexField.setText(" ");
    sailorSsnField.setText(" ");
    buttonPanel.setLayout(gridBagLayout1);
    headerPanel.setLayout(gridBagLayout3);
    xYLayout1.setHeight(460);
    ssnLabel.setText(" SSN");
    ssnLabel.setOpaque(true);
    labelPanel.setLayout(xYLayout3);
    dataPanel.setLayout(xYLayout2);
}

```

```

xYLayout1.setWidth(636);
this.setLayout(xYLayout1);
this.add(headerPanel, new XYConstraints(34, 9, 575, 40));
headerPanel.add(addPrtLabel, new GridBagConstraints2
                                (0, 0, 1, 1, 0.0, 0.0,
                                 GridBagConstraints.WEST,
                                 GridBagConstraints.NONE,
                                 new Insets
                                  (7, 153, 6, 136), 98, 12));
this.add(buttonPanel, new XYConstraints(34, 398, 575, 50));
buttonPanel.add
    (blankPanel, new GridBagConstraints2(3, 0, 1, 1, 1.0, 1.0,
                                         GridBagConstraints.CENTER,
                                         GridBagConstraints.BOTH,
                                         new Insets
                                          (7, 18, 8, 46), 253, 25));
buttonPanel.add(submitButton, new GridBagConstraints2
                (0, 0, 1, 1, 0.0, 0.0,
                 GridBagConstraints.CENTER,
                 GridBagConstraints.NONE,
                 new Insets
                  (15, 8, 12, 0), 0, 0));
buttonPanel.add(clearButton, new GridBagConstraints2
                (1, 0, 1, 1, 0.0, 0.0,
                 GridBagConstraints.CENTER,
                 GridBagConstraints.NONE,
                 new Insets
                  (15, 16, 12, 0), 0, 0));
buttonPanel.add(cancelButton, new GridBagConstraints2
                (2, 0, 1, 1, 0.0, 0.0,
                 GridBagConstraints.CENTER,
                 GridBagConstraints.NONE,
                 new Insets
                  (15, 15, 12, 0), 0, 0));
this.add(dataPanel, new XYConstraints(34, 55, 575, 337));
dataPanel.add(labelPanel, new XYConstraints(8, 12, 138, 316));
labelPanel.add(ssnLabel, new XYConstraints(0, 0, 130, 24));
labelPanel.add(nameLabel, new XYConstraints(0, 32, 130, 24));
labelPanel.add(birthLabel, new XYConstraints(0, 64, 130, 24));
labelPanel.add(sexLabel, new XYConstraints(0, 95, 130, 24));
labelPanel.add(sailorSsnLabel, new XYConstraints(0, 127, 130, 24));
labelPanel.add(jTextField1, new XYConstraints(160, 6, 174, 25));
dataPanel.add(ssnField, new XYConstraints(179, 9, 163, 27));
dataPanel.add(nameField, new XYConstraints(179, 41, 163, 27));
dataPanel.add(birthField, new XYConstraints(179, 73, 163, 27));
dataPanel.add(sexField, new XYConstraints(179, 105, 163, 27));
dataPanel.add(sailorSsnField, new XYConstraints(179, 137, 163, 27));
submitButton.addActionListener(this);
clearButton.addActionListener(this);
cancelButton.addActionListener(this);
}

```

```

/**
 * This method is used perform the actions required based on what the
 * user enters for the database modification.
 *
 * @param      e  action performed by the user
 * @return     void
 * @exception   Default Exception to catch all errors.
 */
public void actionPerformed(ActionEvent e) {
    String command = e.getActionCommand();

    if(command.equals("SUBMIT")){
        Navy.familyStruct  myFamilyStruct = new Navy.familyStruct();
        myFamilyStruct.ssn = ssnField.getText().trim();
        myFamilyStruct.name = nameField.getText().trim();
        myFamilyStruct.sdate = birthField.getText().trim();
        myFamilyStruct.sex = sexField.getText().trim();
        myFamilyStruct.sailorSsn = sailorSsnField.getText().trim();
        parentApplet.corbaBean.addFamily(myFamilyStruct);
    }
    else if(command.equals("CLEAR")){
        ssnField.setText("");
        nameField.setText("");
        birthField.setText("");
        sexField.setText("");
        sailorSsnField.setText("");
    }
    else if(command.equals("CANCEL")){
        parentApplet.showSwitch();
    }
}
}

```

```

//-----
// Filename   : ChangeAddressBean.java
// Authors    : Murat Akbay & Steve Lewis
// Date       : 10/17/1998
// Compiler   : JDK1.1.6 with Symantec JIT Compiler
//-----

import java.awt.*;
import borland.jbcl.layout.*;
import com.sun.java.swing.*;
import java.awt.event.*;
import java.util.*;
import java.io.*;

/**
 * This class is used to create the ChangeAddressBean GUI panel
 *
 * @authors Murat Akbay & Steve Lewis
 */
public class ChangeAddressBean extends JPanel implements
ActionListener{
    XYLayout xYLayout1 = new XYLayout();
    JPanel dataPanel = new JPanel();
    JPanel labelPanel = new JPanel();
    JLabel addressLabel = new JLabel();
    JPanel headerPanel = new JPanel();
    JLabel changeSailorLabel = new JLabel();
    GridBagLayout gridBagLayout3 = new GridBagLayout();
    XYLayout xYLayout2 = new XYLayout();
    JPanel buttonPanel = new JPanel();
    JPanel blankPanel = new JPanel();
    JButton submitButton = new JButton();
    JButton clearButton = new JButton();
    XYLayout xYLayout3 = new XYLayout();
    JLabel cityLabel = new JLabel();
    JLabel stateLabel = new JLabel();
    JLabel countryLabel = new JLabel();
    JLabel postalLabel = new JLabel();
    JLabel homeLabel = new JLabel();
    JLabel sailorLabel = new JLabel();
    JTextField jTextField1 = new JTextField();
    JTextField homePhoneField = new JTextField();
    JTextField sailorSsnField = new JTextField();
    JTextField addressField = new JTextField();
    JTextField cityField = new JTextField();
    JTextField stateField = new JTextField();
    JTextField countryField = new JTextField();
    JTextField postalField = new JTextField();
    JButton cancelButton = new JButton();
    GridBagLayout gridBagLayout1 = new GridBagLayout();
    private Gui parentApplet;

```

```

/**
 * This is the constructor used to create the
 * ChangeAddressBean panel GUI interface
 *
 * @param      Gui parent
 * @return      void
 * @exception   Default Exception to catch all errors.
 */
public ChangeAddressBean(Gui parent) {
    parentApplet = parent;
    try {
        jbInit();
    }
    catch (Exception ex) {
        ex.printStackTrace();
    }
}

/**
 * This method is used to initialize the ChangeAddressBean panel
 *
 * @param      none
 * @return      void
 * @exception   Default Exception to catch all errors.
 */
private void jbInit() throws Exception {
    changeSailorLabel.setText("      CHANGE   SAILOR   ADDRESS");
    submitButton.setText("SUBMIT");
    clearButton.setText("CLEAR");
    cityLabel.setText("  CITY");
    cityLabel.setOpaque(true);
    stateLabel.setText("  STATE");
    stateLabel.setOpaque(true);
    countryLabel.setText("  COUNTRY");
    countryLabel.setOpaque(true);
    postalLabel.setText("  POSTAL  CODE");
    postalLabel.setOpaque(true);
    homeLabel.setText("  HOME  PHONE");
    homeLabel.setOpaque(true);
    sailorLabel.setText("  SAILOR  SSN");
    sailorLabel.setOpaque(true);
    jTextField1.setText(" ");
    homePhoneField.setText(" ");
    sailorSsnField.setText(" ");
    cancelButton.setText("CANCEL");
    addressField.setText(" ");
    cityField.setText(" ");
    stateField.setText(" ");
    countryField.setText(" ");
    postalField.setText(" ");
    buttonPanel.setLayout(gridBagLayout1);
}

```



```

headerPanel.setLayout(gridBagLayout3);
xYLayout1.setHeight(460);
addressLabel.setText("  NEW  ADDRESS");
addressLabel.setOpaque(true);
labelPanel.setLayout(xYLayout3);
dataPanel.setLayout(xYLayout2);
xYLayout1.setWidth(636);
this.setLayout(xYLayout1);
this.add(dataPanel, new XYConstraints(34, 55, 575, 337));
dataPanel.add(labelPanel, new XYConstraints(8, 12, 138, 316));
labelPanel.add(addressLabel, new XYConstraints(0, 0, 130, 24));
labelPanel.add(cityLabel, new XYConstraints(0, 32, 130, 24));
labelPanel.add(stateLabel, new XYConstraints(0, 64, 130, 24));
labelPanel.add(countryLabel, new XYConstraints(0, 95, 130, 24));
labelPanel.add(postalLabel, new XYConstraints(0, 127, 130, 24));
labelPanel.add(homeLabel, new XYConstraints(0, 159, 130, 24));
labelPanel.add(sailorLabel, new XYConstraints(0, 191, 130, 24));
labelPanel.add(jTextField1, new XYConstraints(160, 6, 174, 25));
dataPanel.add(homePhoneField, new XYConstraints(179,168,163,27));
dataPanel.add(sailorSsnField, new XYConstraints(179,200,163,27));
dataPanel.add(addressField, new XYConstraints(179, 9, 163, 27));
dataPanel.add(cityField, new XYConstraints(179, 41, 163, 27));
dataPanel.add(stateField, new XYConstraints(179, 73, 163, 27));
dataPanel.add(countryField, new XYConstraints(179, 105, 163, 27));
dataPanel.add(postalField, new XYConstraints(179, 137, 163, 27));
this.add(headerPanel, new XYConstraints(34, 9, 575, 40));
headerPanel.add(changeSailorLabel, new GridBagConstraints2
                                (0, 0, 1, 1, 0.0, 0.0,
                                 GridBagConstraints.WEST,
                                 GridBagConstraints.NONE,
                                 new Insets
                                  (7, 153, 6, 136), 98, 12));
this.add(buttonPanel, new XYConstraints(34, 398, 575, 50));
buttonPanel.add(blankPanel, new GridBagConstraints2
                                (3, 0, 1, 1, 1.0, 1.0,
                                 GridBagConstraints.CENTER,
                                 GridBagConstraints.BOTH,
                                 new Insets
                                  (7, 18, 8, 46), 253, 25));
buttonPanel.add(submitButton, new GridBagConstraints2
                                (0, 0, 1, 1, 0.0, 0.0,
                                 GridBagConstraints.CENTER,
                                 GridBagConstraints.NONE,
                                 new Insets
                                  (15, 8, 12, 0), 0, 0));
buttonPanel.add(clearButton, new GridBagConstraints2
                                (1, 0, 1, 1, 0.0, 0.0,
                                 GridBagConstraints.CENTER,
                                 GridBagConstraints.NONE,
                                 new Insets
                                  (15, 16, 12, 0), 0, 0));
buttonPanel.add(cancelButton, new GridBagConstraints2

```

```

(2, 0, 1, 1, 0.0, 0.0,
 GridBagConstraints.CENTER,
 GridBagConstraints.NONE,
 new Insets
 (15, 15, 12, 0), 0, 0));

submitButton.addActionListener(this);
clearButton.addActionListener(this);
cancelButton.addActionListener(this);
}

/**
 * This method is used perform the actions required based on what the
 * user enters for the database modification.
 *
 * @param      e  action performed by the user
 * @return     void
 * @exception  Default Exception to catch all errors.
 */
public void actionPerformed(ActionEvent e) {

    String command = e.getActionCommand();

    if(command.equals("SUBMIT")){
        Navy.adressStruct myAddressStruct = new Navy.adressStruct();
        myAddressStruct.address = addressField.getText().trim();
        myAddressStruct.city = cityField.getText().trim();
        myAddressStruct.state = stateField.getText().trim();
        myAddressStruct.country = countryField.getText().trim();
        myAddressStruct.postalCode = postalField.getText().trim();
        myAddressStruct.homePhone = homePhoneField.getText().trim();
        myAddressStruct.ssn = sailorSsnField.getText().trim();
        parentApplet.corbaBean.submitAddressChange(myAddressStruct);
    }
    else if(command.equals("CLEAR")){
        addressField.setText("");
        cityField.setText("");
        stateField.setText("");
        countryField.setText("");
        postalField.setText("");
        homePhoneField.setText("");
        sailorSsnField.setText("");
    }
    else if(command.equals("CANCEL")){
        parentApplet.showSwitch();
    }
}
}

```

```

//-----
// Filename      : GetDaysDeployedBean.java
// Authors       : Murat Akbay & Steve Lewis
// Date          : 10/17/1998
// Compiler      : JDK1.1.6 with Symantec JIT Compiler
//-----

import java.awt.*;
import borland.jbcl.layout.*;
import com.sun.java.swing.*;
import java.awt.event.*;
import java.util.*;

/**
 * This class is used to create the GetDaysDeployedBean GUI panel
 *
 * @authors Murat Akbay & Steve Lewis
 */
public class GetDaysDeployedBean extends JPanel implements
ActionListener{
    XYLayout xYLayout1 = new XYLayout();
    JPanel displayPanel = new JPanel();
    JPanel headerPanel = new JPanel();
    JLabel jLabel8 = new JLabel();
    GridBagLayout gridBagLayout3 = new GridBagLayout();
    GridBagLayout gridBagLayout5 = new GridBagLayout();
    JButton backButton = new JButton();
    JPanel backPanel = new JPanel();
    GridBagLayout gridBagLayout4 = new GridBagLayout();
    JButton clearButton = new JButton();
    JPanel clearPanel = new JPanel();
    GridBagLayout gridBagLayout6 = new GridBagLayout();
    GridBagLayout gridBagLayout7 = new GridBagLayout();
    JButton submitButton = new JButton();
    JPanel submitPanel = new JPanel();
    GridBagLayout gridBagLayout8 = new GridBagLayout();
    JButton cancelButton = new JButton();
    JPanel cancelPanel = new JPanel();
    CardLayout cardLayout1 = new CardLayout();
    GetDaysDeployedPanel1 panel1 ;
    GetDaysDeployedPanel2 panel2 ;
    private Gui parentApplet;

```

```

/**
 * This is the constructor used to create the
 * GetDaysDeployedBean panel GUI interface
 *
 * @param      Gui parent
 * @return     void
 * @exception   Default Exception to catch all errors.
 */
public GetDaysDeployedBean(Gui parent) {
    parentApplet = parent;
    panel1 = new GetDaysDeployedPanel1 (parentApplet);
    try {
        jbInit();
    }
    catch (Exception ex) {
        ex.printStackTrace();
    }
}

/**
 * This method is used to initialize the GetDaysDeployedBean panel
 *
 * @param      none
 * @return     void
 * @exception   Default Exception to catch all errors.
 */
private void jbInit() throws Exception {
    backButton.addActionListener(this);
    clearButton.addActionListener(this);
    submitButton.addActionListener(this);
    cancelButton.addActionListener(this);
    jLabel8.setText("          GET DAYS DEPLOYED FORM");
    backButton.setText("BACK");
    backButton.setOpaque(true);
    clearButton.setText("CLEAR");
    submitButton.setText("SUBMIT");
    submitButton.setOpaque(true);
    cancelButton.setText("CANCEL");
    cancelButton.setOpaque(true);
    displayPanel.setLayout(cardLayout1);
    xYLayout1.setHeight(480);
    displayPanel.add("panel1", panel1);
    cardLayout1.show(displayPanel,"panel1");
    xYLayout1.setWidth(668);
    this.setLayout(xYLayout1);
    this.add(displayPanel, new XYConstraints(38, 49, 606, 362));
    this.add(headerPanel, new XYConstraints(38, 4, 584, -1));
    headerPanel.add(jLabel8, new GridBagConstraints2
                                (0, 0, 1, 1, 0.0, 0.0,
                                 GridBagConstraints.CENTER,
                                 GridBagConstraints.NONE,

```

```

        new Insets
        (5, 157, 8, 132), 98, 12));
this.add(backPanel, new XYConstraints(141, 413, 95, 40));
backPanel.add(backButton, new GridBagConstraints2
        (0, 0, 1, 1, 0.0, 0.0,
        GridBagConstraints.CENTER,
        GridBagConstraints.NONE,
        new Insets
        (0, 0, 0, 0), 0, 0));
this.add(clearPanel, new XYConstraints(246, 413, 95, 40));
clearPanel.add(clearButton, new GridBagConstraints2
        (0, 0, 1, 1, 0.0, 0.0,
        GridBagConstraints.CENTER,
        GridBagConstraints.NONE,
        new Insets
        (0, 0, 0, 0), 0, 0));
this.add(submitPanel, new XYConstraints(346, 413, 95, 40));
submitPanel.add(submitButton, new GridBagConstraints2
        (0, 0, 1, 1, 0.0, 0.0,
        GridBagConstraints.CENTER,
        GridBagConstraints.NONE,
        new Insets
        (0, 0, 0, 0), 0, 0));
this.add(cancelPanel, new XYConstraints(449, 413, 95, 40));
cancelPanel.add(cancelButton, new GridBagConstraints2
        (0, 0, 1, 1, 0.0, 0.0,
        GridBagConstraints.CENTER,
        GridBagConstraints.NONE,
        new Insets
        (0, 0, 0, 0), 0, 0));

}

/**
 * This method is used perform the actions required based on what the
 * user enters for the database modification.
 *
 * @param      e  action performed by the user
 * @return     void
 * @exception  Default Exception to catch all errors.
 */
public void actionPerformed(ActionEvent e) {

    String command = e.getActionCommand();

    if(command.equals("BACK")){
        cardLayout1.show(displayPanel, "panel1");
    }
    else if(command.equals("SUBMIT")){
        String[] entryString = panel1.getInputs();

```

```

String[] columnNames = {"Tango No.", "Date Of Departure",
                        "Rank Rate", "F. Name",
                        "L. Name", "Command Name",
                        "TAD Title", "TAD Date Of Arrival",
                        "TAD Date Of Departure",
                        "Destination Command"};

Navy.deploymentStruct[] result =
    parentApplet.corbaBean.getDaysDeployed
        (entryString[0], entryString[1],
         entryString[2], entryString[3]);

if( result == null ){
    JOptionPane.showMessageDialog
        (parentApplet, "Error in executing query", "",
         JOptionPane.ERROR_MESSAGE );
}
int resultLength = result.length;
if( resultLength == 0 ){
    JOptionPane.showMessageDialog
        (parentApplet, "The query returned no results"
         , "", JOptionPane.INFORMATION_MESSAGE);
}
else{
    Object[][] QueryResult = new Object[resultLength][10];
    for(int ix = 0; ix < resultLength ; ix++ ){
        QueryResult[ix][0] = result[ix].tangoNumber;
        QueryResult[ix][1] = result[ix].dateDepart;
        QueryResult[ix][2] = result[ix].rankRate;
        QueryResult[ix][3] = result[ix].firstName;
        QueryResult[ix][4] = result[ix].lastName;
        QueryResult[ix][5] = result[ix].commandName;
        QueryResult[ix][6] = result[ix].title;
        QueryResult[ix][7] = result[ix].dateArrival;
        QueryResult[ix][8] = result[ix].dateDeparture;
        QueryResult[ix][9] = result[ix].comm;
    }
    panel2 = new GetDaysDeployedPanel2
        (parentApplet, QueryResult, columnNames);
    displayPanel.add("panel2", panel2);
    cardLayout1.show(displayPanel, "panel2");
}
}
else if(command.equals("CLEAR")){
    panel1.clear();
}
else if(command.equals("CANCEL")){
    parentApplet.showSwitch();
}
}
}

```

```

//-----
// Filename    : GetDaysDeployedPanel1.java
// Authors     : Murat Akbay & Steve Lewis
// Date        : 10/17/1998
// Compiler    : JDK1.1.6 with Symantec JIT Compiler
//-----

import java.awt.*;
import borland.jbcl.layout.*;
import com.sun.java.swing.*;
import java.awt.event.*;

/**
 * This class is used to create the GetDaysDeployedPanel1 GUI panel
 *
 * @authors Murat Akbay & Steve Lewis
 */
public class GetDaysDeployedPanel1 extends JPanel {
    private String sample = "Sample";
    XYLayout xYLayout1 = new XYLayout();
    JPanel jPanel1 = new JPanel();
    JPanel jPanel2 = new JPanel();
    JLabel nameLabel = new JLabel();
    JLabel sDateLabel = new JLabel();
    JTextField lastNameField = new JTextField();
    JTextField startDateField = new JTextField();
    XYLayout xYLayout2 = new XYLayout();
    XYLayout xYLayout3 = new XYLayout();
    JLabel jLabel1 = new JLabel();
    private Gui parentApplet;
    JLabel ssnLabel = new JLabel();
    JLabel eDateLabel = new JLabel();
    JTextField stopDateField = new JTextField();
    JTextField ssnField = new JTextField();
    JLabel jLabel2 = new JLabel();

    /**
     * This is the constructor used to create the
     * GetDaysDeployedPanel1 panel GUI interface
     *
     * @param      Gui parent
     * @return     void
     * @exception   Default Exception to catch all errors.
     */
    public GetDaysDeployedPanel1 (Gui parent) {
        parentApplet = parent;
        try {
            jbInit();
        }
        catch (Exception ex) {

```

```

        ex.printStackTrace();
    }
}

/**
 * This method is used to initialize the GetDaysDeployedPanel1 panel
 *
 * @param      none
 * @return     void
 * @exception   Default Exception to catch all errors.
 */
private void jbInit() throws Exception {
    jLabel1.setText
        ("Enter the sailor\'s name or social security number and the
         time interval ");
    ssnLabel.setOpaque(true);
    eDateLabel.setOpaque(true);
    stopDateField.setText(" ");
    ssnField.setText(" ");
    jLabel2.setText("and click on the submit button.");
    eDateLabel.setText(" STOP DATE");
    ssnLabel.setText(" SAILOR SSN");
    xYLayout1.setHeight(360);
    NameLabel.setOpaque(true);
    sDateLabel.setOpaque(true);
    lastNameField.setText(" ");
    startDateField.setText(" ");
    sDateLabel.setText(" START DATE");
    NameLabel.setText(" LAST NAME");
    jPanel2.setLayout(xYLayout3);
    jPanel1.setLayout(xYLayout2);
    xYLayout1.setWidth(606);
    this.setLayout(xYLayout1);
    this.add(jPanel1, new XYConstraints(31, 1, 574, 359));
    jPanel1.add(jPanel2, new XYConstraints(-3, 5, 189, 198));
    jPanel2.add(NameLabel, new XYConstraints(25, 99, 130, 26));
    jPanel2.add(sDateLabel, new XYConstraints(25, 138, 130, 26));
    jPanel2.add(ssnLabel, new XYConstraints(25, 60, 130, 26));
    jPanel2.add(eDateLabel, new XYConstraints(25, 177, 130, 26));
    jPanel1.add(lastNameField, new XYConstraints(189, 105, 163, 27));
    jPanel1.add(startDateField, new XYConstraints(189, 141, 163, 27));
    jPanel1.add(jLabel1, new XYConstraints(11, 215, 501, 48));
    jPanel1.add(stopDateField, new XYConstraints(189, 177, 163, 27));
    jPanel1.add(ssnField, new XYConstraints(189, 69, 163, 27));
    jPanel1.add(jLabel2, new XYConstraints(11, 264, 498, 44));
}

```



```

/**
 * This method is used to take the inputs that the user enters
 * in the GetDaysDeployedPanel1 panel
 *
 * @param      none
 * @return     replyString which contains the user inputs
 * @exception  none
 */
public String[] getInputs(){
    String[] replyString = new String[4];
    replyString[0] = ssnField.getText().trim();
    replyString[1] = lastNameField.getText().trim();
    replyString[2] = startDateField.getText().trim();
    replyString[3] = stopDateField.getText().trim();
    return replyString;
}

/**
 * This method is used to clear the fields on the
 * GetDaysDeployedPanel1 panel
 *
 * @param      none
 * @return     void
 * @exception  none
 */
public void clear() {
    ssnField.setText("");
    lastNameField.setText("");
    startDateField.setText("");
    stopDateField.setText("");
}
}

```

```

//-----
// Filename   : GetDaysDeployedPanel2.java
// Authors    : Murat Akbay & Steve Lewis
// Date       : 10/17/1998
// Compiler   : JDK1.1.6 with Symantec JIT Compiler
//-----

import java.awt.*;
import borland.jbcl.layout.*;
import com.sun.java.swing.*;
import com.sun.java.swing.table.*;
import java.awt.event.*;
import borland.dbswing.*;

/**
 * This class is used to display the data returned from the user
 * defined query submitted through the GetDaysDeployedPanel2 panel
 *
 * @authors Murat Akbay & Steve Lewis
 */
public class GetDaysDeployedPanel2 extends JPanel {

    BorderLayout borderLayout1 = new BorderLayout();
    private Gui parentApplet;

    /**
     * This constructor is used to create the GetDaysDeployedPanel1 GUI
     * panel
     *
     * @params Gui parent, Object [][] inData, String[] inColumnNames
     * @returns none
     * @exceptions none
     */
    public GetDaysDeployedPanel2
        (Gui parent, Object [][] inData, String[] inColumnNames) {
        parentApplet = parent;
        this.setLayout(borderLayout1);
        final JTable table = new JTable(inData, inColumnNames);
        table.setPreferredScrollableViewportSize(new Dimension(600, 360));
        table.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);
        TableColumn column = null;
        for (int i = 0; i < 10; i++) {
            column = table.getColumnModel().getColumn(i);
            column.setMinWidth(100);
        }
        add(JTable.createScrollPaneForTable(table), BorderLayout.CENTER);
    }
}

```

```
//-----
// Filename      : Gui.html
// Authors       : Murat Akbay & Steve Lewis
// Date          : 10/17/1998
// Compiler      : JDK1.1.6 with Symantec JIT Compiler
//-----

<!-- This html file is downloaded to the client web-browser and is used
      by the browser to create the applet which will serve as the client
      front-end to the EIS.  This will work with I.E. 4.0+ and Netscape
      4.5+ web-browsers-->

<HTML>
<HEAD>
<TITLE> CORBAWORKS </TITLE>
<BODY BACKGROUND="spark&quill.gif">
<P>

<!-- The following code is specified at the beginning of the <BODY>
tag. -->
<CENTER>
<OBJECT classid="clsid:8AD9C840-044E-11D1-B3E9-00805F499D93"
width="790" height="596"
codebase="http://java.sun.com/products/plugin/1.1/
          jinstall-11-win32.cab#Version=1,1,0,0">
  <PARAM NAME="code" VALUE="Gui.class">
  <PARAM NAME="type" VALUE="application/x-java-applet;version=1.1">
  <PARAM NAME="org.omg.CORBA.ORBClass"
          VALUE="com.visigenic.vbroker.orb.ORB">
  <PARAM NAME=ORBservices VALUE=CosNaming>
  <PARAM NAME=SVCnameroot VALUE=JavaCorba>
</CENTER>
  <COMMENT>
    <EMBED type="application/x-java-applet;version=1.1"
          WIDTH="790" HEIGHT="596"
          HSPACE="100" ALIGN="Top" CODE="Gui.class"
          pluginspage="http://java.sun.com/products/plugin/1.1/
                      plugin-install.html
                      org.omg.CORBA.ORBClass="com.visigenic.vbroker.orb.ORB"
                      ORBservices=CosNaming
                      SVCnameroot=JavaCorba>
  </NOEMBED>
</COMMENT>
    No JDK 1.1 support for APPLET!!
  </NOEMBED></EMBED>
</OBJECT>

</P>
</BODY>
</HTML>
```

```
//-----
// Filename   : Guijar.html
// Authors    : Murat Akbay & Steve Lewis
// Date       : 10/17/1998
// Compiler   : JDK1.1.6 with Symantec JIT Compiler
//-----
```

```
<!-- This html file is downloaded to the client web-browser and is used
by the browser to create the applet which will serve as the client
front-end to the EIS. This will work with I.E. 4.0+ and Netscape
4.5+ web-browsers. If this file is used by the browser, additional
savings of download time will occur, since the files that make up
the applet are downloaded in a single session.-->
```

```
<HTML>
<HEAD>
<TITLE> CORBAWORKS </TITLE>
<BODY BACKGROUND="spark&quill.gif">
<P>
<!-- The following code is specified at the beginning of the <BODY>
tag. -->
<CENTER>
<OBJECT classid="clsid:8AD9C840-044E-11D1-B3E9-00805F499D93"
width="775" height="596"
codebase="http://java.sun.com/products/plugin/1.1/
jinstall-11-win32.cab#Version=1,1,0,0">
<PARAM NAME="code" VALUE="Gui.class">
<PARAM NAME="archive" VALUE="Gui.jar">
<PARAM NAME="type" VALUE="application/x-java-applet;version=1.1">
<PARAM NAME="org.omg.CORBA.ORBClass"
VALUE="com.visigenic.vbroker.orb.ORB">
<PARAM NAME="ORBservices" VALUE="CosNaming">
<PARAM NAME="SVCnameroot" VALUE="JavaCorba">
</CENTER>
<COMMENT>
<EMBED type="application/x-java-applet;version=1.1"
WIDTH="775" HEIGHT="596"
HSPACE="100" ALIGN="Top" CODE="Gui.class"
ARCHIVE="Gui.jar"
pluginspage="http://java.sun.com/products/plugin/1.1/
plugin-install.html
org.omg.CORBA.ORBClass="com.visigenic.vbroker.orb.ORB"
ORBservices=CosNaming
SVCnameroot=JavaCorba>
<NOEMBED>
</COMMENT>
No JDK 1.1 support for APPLETT!!
</NOEMBED></EMBED>
</OBJECT>
</P>
</BODY>
</HTML>
```

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center2
8725 John J. Kingman Road, Ste 0944
Ft. Belvoir, VA 22060-6218

2. Genelkurmay Baskanligi1
Personel Baskanligi
Bakanliklar
Ankara, TURKEY

3. Kara Kuvvetleri Komutanligi1
Personel Daire Baskanligi
Bakanliklar
Ankara, TURKEY

4. Kara Kuvvetleri Komutanligi1
Kutuphanesi
Bakanliklar
Ankara, TURKEY

5. Kara Harp Okulu2
Kutuphanesi
Dikmen
Ankara, TURKEY

6. Dudley Knox Library2
Naval Postgraduate School
411 Dyer Rd.
Monterey, CA 93943-5101

7. Commanding Officer1
(Attn: Code 30, CDR Zellman)
Naval Information Warfare Activity
9800 Savage Rd.
Ft Meade, MD 20755-6000

8. Commanding Officer1
(Attn: Code 00, CDR Arbogast)
Naval Security Group Activity, Pensacola Florida
475 Jones Street
NTTC Corry Station
Pensacola, FL 32511

9. Commander Naval Security Group Command1
Naval Security Group Headquarters
9800 Savage Road, Suite 6585
Fort George G. Meade, MD 20755-6585
10. Chairman, Code CS1
Naval Postgraduate School
Monterey, CA 93943-5101
11. Dr. C. Thomas Wu, Code CS/Wu.....1
Naval Postgraduate School
Monterey, CA 93943-5100
12. LCDR Chris Eagle, Code EC/Lt.....1
Naval Postgraduate School
Monterey, CA 93943-5100
13. 1LT Murat Akbay1
218/8 Sokak Emek Apt. No. 19
Bornova, Izmir, TURKEY
14. LT Steven Lewis1
4921 E. 138th St SW
Edmonds, WA 98026